Desarrollo del aplicativo en python basado en la “Guía para el cálculo y la selección de apoyos en estructuras de media y baja tensión para sistemas de distribución actualizado a normatividad vigente, RETIE y NTC 2050”

Brayan Manuel Quiroga Luque, y Juan Pablo Sanmiguel Diaz

Trabajo de grado para optar por el título de

Ingeniero Electricista

Director

Rolando Andrés Rincón Saravia

Especialista en Gerencia de Proyectos

Codirector

Oscar Arnulfo Quiroga Quiroga

Doctor en Ingeniería Eléctrica

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones

Ingeniería eléctrica

Bucaramanga

2024

Dedicatoria

Quiero dedicar este importante logro en mi vida a mis padres, Manuel y Milena, quienes, con gran sacrificio, amor y un apoyo incondicional han sido los pilares que me han guiado, transmitiéndome principios y valores conduciéndome a ser la persona que soy hoy en día. A mi hermano Marlon, cuya complicidad y aliento han sido como un faro en las etapas desafiantes, compartiendo risas y desafíos que han fortalecido nuestros lazos. A mi abuela Lilia, cuyas enseñanzas y amor han dejado una marca imborrable en mi camino académico. A Carolina, mi pareja, agradezco profundamente tu apoyo constante y tu comprensión han sido fundamentales en este viaje. Este logro también es tuyo, y estoy agradecido por compartir cada paso de este camino contigo. A todos ustedes, mi familia y seres queridos, les dedico este logro con gratitud y amor. Su presencia ha sido mi mayor fortaleza y motivación a lo largo de esta travesía académica

-Brayan

Quiero expresar mi profundo agradecimiento a varias personas que han sido fundamentales en el camino hacia la culminación de mi tesis de grado. En primer lugar, quiero dedicar unas palabras de gratitud a mis padres, Sara y Rodrigo, cuyo amor, sacrificio y apoyo incondicional han sido el pilar de mi trayectoria académica. Les debo este logro y cada uno de mis éxitos. A mi hermano que ha sido una parte esencial de mi vida, brindándome apoyo en cada paso de mi infancia y crecimiento profesional. Su constante aliento ha sido una fuente de inspiración. Agradezco sinceramente a mis amigos y compañeros de estudio, Freily, Johan, Brayan y Juan Sebastián. Su amistad y colaboración han sido cruciales durante esta etapa tan significativa de mi vida. Juntos hemos compartido experiencias y crecido mutuamente, convirtiendo esta etapa académica en una experiencia inolvidable.

-Juan

Agradecimientos

Agradecemos a nuestro director Rolando Rincón y codirector de tesis Oscar Quiroga. Su liderazgo, paciencia y colaboración fueron fundamentales en el desarrollo y éxito de este proyecto. A través de su guía, hemos adquirido conocimientos valiosos que trascienden la esfera académica.

Agradecemos a todos nuestros compañeros y profesores de la E3T que han contribuido tanto personal como profesionalmente, en nuestra vida. Cada interacción ha dejado una huella imborrable y ha enriquecido nuestro camino hacia el logro de esta meta académica. A todos ustedes, nuestros más sinceros agradecimientos.

Tabla de Contenido

Pág.

Lista de Figuras

Pág.

Resumen

Título: Desarrollo del aplicativo en Python basado en la “guía para el cálculo y la selección de apoyos en estructuras de media y baja tensión para sistemas de distribución actualizado a normatividad vigente, RETIE y NTC 2050” *

Autores: Brayan Manuel Quiroga Luque, Juan Pablo Sanmiguel Diaz **

Palabras clave: Cálculos mecánicos, NTC 2050, RETIE, Software de diseño, Estructuras de concreto

Descripción: Vivimos en una era donde lo digital se ha hecho una necesidad diaria, siendo la programación una habilidad fundamental que impulsa la innovación y la resolución de problemas en una amplia gama de disciplinas. Entre los numerosos lenguajes de programación disponibles, Python ha emergido como uno de los más destacados y versátiles. Su popularidad creciente se debe en gran parte a su simplicidad, legibilidad y amplio conjunto de bibliotecas, que lo convierten en una herramienta invaluable para programadores, científicos de datos, ingenieros de software y muchas otras profesiones relacionadas con la informática.

Se seleccionó Python como lenguaje de programación principal por su conocida legibilidad y sencillez, que facilitan la realización de complicados algoritmos necesarios para los cálculos mecánicos en estructuras de distribución. Además, la metodología Scrum y la programación orientada a objetos aportan estructura y agilidad al desarrollo del software, permitiendo modularidad, reutilización y adaptación a medida que avanza el proyecto.

Este software está diseñado específicamente para realizar cálculos mecánicos de estructuras de concreto de media tensión y sería accesible de forma gratuita. De esta manera, los usuarios pueden aprovechar esta herramienta sin incurrir en altos costos de licencias. Además, el desarrollo del software está basado en la "Guía para el cálculo y la selección de apoyos en estructuras de media y baja tensión para sistemas de distribución actualizado a normatividad vigente, RETIE y NTC2050”, lo que garantizó la precisión y la coherencia en los resultados obtenidos.

Se prevé que esta herramienta contribuya sustancialmente a la mejora de la eficiencia y la calidad en el diseño de sistemas eléctricos de media tensión, así como a la accesibilidad de la tecnología, al ofrecer una solución tecnológica basada en Python, enfocada a la programación orientada a objetos y gestionada con Scrum.

* Trabajo de grado.

**Facultad de Ingenierías Físicomecánicas. Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones. Director: Rolando Andrés Rincón Saravia. Especialista en Gerencia de Proyectos. Codirector: Oscar Arnulfo Quiroga Quiroga. Doctor en Ingeniería Eléctrica

Abstract

Title: Development of the Python Application based on the "Guide for the Calculation and Selection of Supports in Medium and Low Voltage Structures for Distribution Systems updated to current regulations, RETIE, and NTC 2050" *

Authors: Brayan Manuel Quiroga Luque, Juan Pablo Sanmiguel Diaz **

Keywords: Mechanical calculations, NTC 2050, RETIE, Design software, Concrete structures

Description: We live in an era where digital has become a daily necessity, with programming being a fundamental skill driving innovation and problem-solving in a wide range of disciplines. Among the numerous programming languages available, Python has emerged as one of the most prominent and versatile. Its growing popularity is largely due to its simplicity, readability, and extensive library set, making it an invaluable tool for programmers, data scientists, software engineers, and many other computer-related professions.

Python was selected as the primary programming language for its well-known readability and simplicity, facilitating the implementation of complex algorithms necessary for mechanical calculations in distribution structures. Additionally, the Scrum methodology and object-oriented programming contribute structure and agility to the software development, allowing modularity, reusability, and adaptation as the project progresses.

This software is specifically designed for performing mechanical calculations of medium-voltage concrete structures and would be accessible for free. In this way, users can leverage this tool without incurring high license costs. Furthermore, the software development is based on the "Guide for the Calculation and Selection of Supports in Medium and Low Voltage Structures for Distribution Systems updated to current regulations, RETIE, and NTC2050," ensuring accuracy and consistency in the obtained results.

It is anticipated that this tool will significantly contribute to improving efficiency and quality in the design of medium-voltage electrical systems, as well as technology accessibility by providing a technological solution based on Python, focused on object-oriented programming, and managed with Scrum.

*Degree Work.

**Faculty of Physical-Mechanical Engineering. School of Electrical, Electronic and Telecommunications Engineering. Director: Rolando Andres Rincón Saravia. Specialist in project managing. Co-director: Óscar Arnulfo Quiroga Quiroga. PhD in Electrical Engineering

INTRODUCCIÓN

El diseño y la implementación de sistemas eléctricos confiables en media tensión requieren cálculos mecánicos precisos y una cuidadosa selección de estructuras. Estos aspectos son fundamentales para garantizar la seguridad y eficiencia eléctrica en proyectos de ingeniería. Sin embargo, el acceso a herramientas especializadas a menudo es costoso y restringido, limitando la capacidad de profesionales y estudiantes para realizar análisis detallados. En respuesta a esta necesidad, este trabajo de grado se propone desarrollar un software basado en Python para realizar cálculos mecánicos en estructuras de concreto de media tensión, siguiendo la normativa actual y la utilización de la programación orientada a objetos junto con la implementación de una metodología de proyectos eficiente como es la metodología scrum. Con este enfoque se busca superar las limitaciones de costos y acceso, brindando una solución eficiente y asequible para el diseño en ingeniería eléctrica.

En el primer capítulo para obtener una comprensión básica del lenguaje, se iniciará con una breve sinopsis de Python y con ayuda de una serie de datos obtener una visión general, más adelante examinaremos sus características principales, su organización, la información fundamental y otros elementos, así mismo, presentaremos la programación orientada a objetos y su implementación, igualmente, el enfoque de gestión de proyectos Scrum será una pieza clave que describiremos, y por último se realizará una breve caracterización de la “Guía para el cálculo y la selección de apoyos en estructuras de media y baja tensión para sistemas de distribución actualizado a normatividad vigente, RETIE y NTC 2050”. En el capítulo 2, se aborda exhaustivamente el proceso de desarrollo del software propuesto, destacando cada etapa crucial que contribuyó a su creación y funcionalidad. Brindando una visión detallada del proceso de desarrollo, desde la concepción de la solución hasta su implementación práctica.

1. Marco de referencia

1.1. Programación en Python

1.1.1. ¿Qué es Python?

Python es un lenguaje de programación que a lo largo de los últimos años ha venido en un crecimiento constante. Actualmente, siendo uno de los lenguajes de programación más completos que es implementado al momento del desarrollo de software debido a su potencia, flexibilidad y sintaxis clara. No requiere tiempo de compilación, ya que es un lenguaje interpretado, lo que lo convierte en un lenguaje de alto nivel.

Además, Python es de código abierto, esto significa que su desarrollo y difusión están abiertos a cualquier persona interesada. No requiere una licencia de pago para distribuir software desarrollado con este lenguaje.

1.1.2. Principales características

Python es reconocido por su legibilidad y su cercanía al lenguaje humano, lo que lo hace adecuado para programadores novatos. Es un lenguaje interpretado, lo que significa que no necesita una etapa de compilación, esto facilita el desarrollo y la depuración. Es un lenguaje multiparadigma, que se puede programar de diferentes maneras, incluyendo orientación a objetos, programación imperativa y funcional. Utiliza un sistema de tipado dinámico, eliminando la necesidad de declarar el tipo de variable, dado que este se define durante la ejecución del programa.

La gestión automática de memoria es una característica importante de Python, ya que un recolector de basura se encarga de asignar y liberar memoria de forma automática, minimizando la posibilidad de errores de acceso a la memoria. Ofrece una biblioteca estándar amplia que contiene módulos y funciones para diversas tareas, desde manipulación de archivos hasta desarrollo web. Además, es altamente portable, lo que significa que los

programas escritos en Python son ejecutables en diferentes tipos de plataformas y sistemas operativos sin cambios significativos, como Windows, macOS y sistemas basados en Unix.

La comunidad activa de desarrolladores de Python contribuye con bibliotecas y documentación, lo que simplifica el desarrollo y la solución de problemas. Python pone énfasis en la legibilidad del código, lo que facilita su mantenimiento y la colaboración en proyectos. Además, sigue un enfoque orientado a objetos, lo que implica que cada elemento en él es considerado como un objeto que posee atributos y métodos.

Python tiene soporte para programación web a través de frameworks y bibliotecas como Django y Flask. Además, se utiliza ampliamente en ciencia de datos y análisis gracias a bibliotecas como NumPy, pandas, Matplotlib y scikit-learn. Se integra fácilmente con otros lenguajes, como C, C++ y Java, lo que permite extender aplicaciones y aprovechar bibliotecas existentes. Estas características hacen que Python sea un lenguaje versátil y muy popular en el mundo de la programación.

1.1.3. Herramientas de desarrollo

Uno de los aspectos significativos a considerar al momento del desarrollo de un software es el grupo de recursos disponibles para llevar a cabo la tarea. Más allá de la tecnología o el lenguaje específico, Python posee diversos tipos de utilidades para el desarrollo de un software, como lo son los editores de texto simples hasta herramientas de depuración complejas, junto con entornos de desarrollo integrados que proporcionan múltiples funciones en una sola aplicación. En el caso de Python, no es diferente; el lenguaje dispone de una variedad de herramientas de desarrollo que aumentarán nuestra productividad.

Dado que explorar en detalle todas y cada una de las herramientas de desarrollo disponibles para la creación de código con Python está fuera del alcance de este libro, nos encaminaremos por las más populares y con las cuales se va a realizar este proyecto.

1.1.4. Conceptos básicos

En Python, un programa típicamente se compone de varios elementos. El lenguaje nos proporciona varios tipos diferenciados de estos elementos. El primero es el objeto, podemos describirlo como un elemento que reside en la memoria y que está vinculado a un conjunto de valores y operaciones que pueden llevarse a cabo con él. Las expresiones, por su parte, se conforman mediante la combinación de operadores, constantes, valores, variables, y funciones, las cuales se emplean conforme a reglas particulares. Estas expresiones generalmente se agrupan para formar sentencias, que son consideradas las unidades ejecutables más pequeñas de un programa. Por último, tenemos los módulos, son componentes clave para organizar, reutilizar y estructurar el código de manera efectiva, lo que facilita el desarrollo y el mantenimiento de programas.

Python viene con una serie de objetos incorporados para facilitar la programación. Proporcionan una serie de ventajas, como un rendimiento excelente con un uso mínimo de memoria durante la ejecución, facilidad para desarrollar estructuras complicadas basadas en ellos y la disminución de tiempo al no tener que fabricar manualmente estas estructuras de datos. Python ofrece específicamente tuplas, conjuntos, diccionarios, números, cadenas, booleanos, listas y archivos. Además, se emplea un tipo de objeto único conocido como "None" para designar un valor nulo.

1.1.5. Operadores

Bajo el contexto del trabajo con números en Python, es lógico pensar en la interacción entre ellos, en otras palabras, la realización de operaciones numéricas. Este lenguaje de

programación nos proporciona una variedad de operadores aritméticos con el fin de que podamos llevar a cabo la realización de nuestro código. Además, para operaciones a nivel de bits, tenemos a nuestra disposición operaciones como NOT, NOR, XOR y AND. Igualmente se cuenta con operadores para verificar igualdades y desigualdades, así como para ejecutar operaciones lógicas como OR y AND. Las sintaxis de estas operaciones están recolectadas en **la tabla 1.**

Al igual que en otros lenguajes de programación, Python sigue reglas de precedencia de operadores, lo que requiere que tengamos en cuenta al escribir expresiones que involucran varios de ellos. Además, es importante recordar que los paréntesis pueden utilizarse para establecer la prioridad de unas operaciones sobre otras dentro de una misma expresión.

Figura 1

Operaciones y operadores en python

Operación	**Sintaxis**
Suma	$x + y$
Resta	$x - y$
Multiplicación	*x*y*
División	*x/y*
División Entera	*x//y*
Potencia	*x**y*
OR (bit)	*a|b*
XOR (bit)	*x^y*
AND (bit)	*x&y*
Igual	*x==y*
Desigual	*x!=y*
OR (lógica)	*X or b*
AND (lógica)	*A and b*
Negación (lógica)	*not a*

1.1.6. Vectores

En Python, los vectores se pueden representar de varias maneras, pero una de las formas más comunes es utilizando listas.

1.1.6.1 Operaciones con vectores.

- **Suma de vectores:** Puedes sumar dos vectores componente por componente simplemente sumando las listas correspondientes.
- **Resta de vectores**: La resta de vectores se realiza de manera similar a la suma, restando las listas componente por componente.
- **Producto escalar:** El producto escalar de vectores se da multiplicando los componentes alojados en cada vector y luego sumando los resultados.
- **Producto cruz:** El producto cruz entre dos vectores se calcula con una librería extra siendo esta NumPy.

1.1.7. Matrices

En Python, las matrices se pueden representar de varias maneras, pero una de las formas más comunes de hacerlo es utilizando listas anidadas, donde cada lista interna representa una fila de la matriz.

Puedes crear una matriz utilizando listas anidadas. Cada lista interna representa una fila de la matriz. Puedes acceder a elementos específicos de una matriz utilizando índices.

1.1.8. Sentencias de control

Aprender las sentencias de control es un primer paso crucial en el aprendizaje de cualquier lenguaje de programación. Las sentencias fundamentales en el lenguaje Python permiten formular condiciones y realizar iteraciones.

Así como otros lenguajes de programación, Python tiene una colección de sentencias de control. Entre ellas se incluyen algunas de las más fundamentales y utilizadas que están presentes en otros lenguajes, como for, while y if/else, así como otras más especializadas como pass y with.

1.1.8.1 If, Else y Elif. Las estructuras if, else y elif de Python son cruciales para la toma de decisiones bajo condiciones. Cuando se cumple una condición particular, el bloque if permite la ejecución de una secuencia de instrucciones. El bloque de código indentado bajo if se ejecuta una vez que se ha evaluado esta condición y se ha determinado que es cierta.

La sentencia else se utiliza cuando la condición if es falsa y es necesario un curso de acción diferente. En el caso de que la condición if inicial sea falsa, se utiliza para llevar a cabo un conjunto alternativo de instrucciones. Cuando la condición no se cumple, se crea un camino alternativo en el flujo del programa.

Además, la sentencia elif (abreviatura de "else if") se emplea cuando hay varias condiciones que deben evaluarse en orden. Esto permite verificar más condiciones antes de una sentencia else pero después de una sentencia if. Elif es útil en situaciones en las que hay varias condiciones y es necesaria una verificación secuencial, ya que puede evaluar y ejecutar bloques de código en función de la primera condición verdadera descubierta, omitiendo las comprobaciones restantes.

Estas estructuras son esenciales para gestionar flujos de programas con condiciones, que permiten a un programa adaptarse dinámicamente a diversas situaciones. La lógica de programación se basa en la capacidad de tomar decisiones en función de las condiciones, lo que permite crear aplicaciones capaces de reaccionar y adaptarse a diversas circunstancias.

1.1.8.2 For y While. Las líneas “for” y “while” nos permiten crear bucles más fácilmente y mejorar la ejecución de las iteraciones. Para cada elemento que compone un objeto aplicado, se realiza una secuencia de instrucciones mediante la sentencia “for”. El método “range()” de Python es una herramienta útil para iterar a través de una lista de valores.

La sentencia "for" tiene un bloque "else" opcional que puede incluirse. Si esta declaración está presente, el bucle terminará con todas las instrucciones consecutivas

ejecutadas, a menos que se active otra declaración, como "break", para terminar el bucle antes de tiempo. Por ejemplo, cuando se ejecuta un bucle "for" sin una declaración "break", las instrucciones del bloque "else" siempre se ejecutarán cuando finalice el bucle.

Otra sentencia que se emplea para la iteración es "while". Esta sentencia ejecuta un conjunto de instrucciones mientras una condición o conjunto de condiciones particulares se mantengan ciertas. Para finalizar la ejecución del bucle, existen varias técnicas disponibles. La forma más simple consiste en modificar la condición o las condiciones iniciales de manera que ya no se cumplan, lo que detendrá la iteración. Otra alternativa es utilizar la sentencia "break" directamente, lo que resultará en una salida inmediata del bucle. Es importante destacar que esta última sentencia también es válida para su uso en bucles "for".

1.1.9. Funciones

Las funciones son uno de los elementos centrales en el marco de la programación estructurada. Una función es una colección de comandos que pueden llamarse o llevarse a cabo repetidamente mientras se ejecuta un programa. El uso de funciones tiene claras ventajas, como menos código en general, mejor legibilidad del código y más posibilidades de reutilización del código.

1.1.9.1 Definir una función. Puedes definir una función en Python utilizando la palabra clave “def”, seguida del nombre de la función y paréntesis que pueden contener parámetros. Después del encabezado de la función, el código de la función se define con sangría.

1.1.9.2 Llamada a una función. Una vez que hayas definido una función, puedes llamarla para ejecutar el código dentro de ella. Para llamar una función, escribe su nombre seguido de paréntesis, y puedes pasar argumentos a la función si esta los requiere.

1.1.9.3 Parámetros de una función. Los parámetros son valores que se pasan a la

función cuando se llama. Las funciones pueden tener cero o más parámetros. En el ejemplo anterior, la función suma toma dos parámetros, a y b, que se utilizan en el cálculo de la suma. Los parámetros son variables locales dentro de la función y solo existen dentro del alcance de la función.

1.1.9.4 Valor de retorno. La palabra clave "return" permite a las funciones devolver un valor. El código que invoca la función puede hacer uso del valor devuelto. La función "return resultado" se utiliza en el ejemplo de la función "suma" para recuperar el resultado de la suma.

1.1.9.5 Funciones sin Valor de Retorno. No todas las funciones necesitan devolver un valor. Algunas funciones se utilizan para realizar una tarea sin necesidad de devolver un resultado. En este caso, la función puede no tener una declaración return o puede tener return sin un valor.

1.1.9.6 Argumentos por Defecto. La función puede invocarse con menos argumentos si se establecen valores por defecto para sus parámetros. Si no se proporciona ningún valor al invocar la función, se aplican los valores por defecto indicados en la especificación de la función.

1.1.9.7 Ámbito de Variables. Las variables definidas dentro de una función son locales a esa función y no pueden accederse desde fuera de la función, a menos que se devuelvan mediante return. Por lo tanto, las variables definidas dentro de una función no interfieren con variables del mismo nombre en otros lugares del programa.

1.1.9.8 Funciones Incorporadas. Python proporciona una amplia gama de funciones incorporadas que están disponibles sin necesidad de importar módulos adicionales. Algunas de las funciones incorporadas más comunes en Python son las siguientes:

- $print()$: Utilizada para imprimir valores en la consola.

- $len()$: Calcula la longitud de una secuencia, como una lista o una cadena.
- $input()$: Permite al usuario ingresar datos desde la consola.
- $str(), int(), float()$: Se utilizan para convertir valores a cadenas, enteros o números de punto flotante, respectivamente.
- $round()$: Redondea un número al entero más cercano.
- $enumerate()$: Proporciona índices y valores de una secuencia al mismo tiempo.
- $zip()$: Combina varias secuencias en una secuencia de tuplas.
- $any(), all()$: Verifican si al menos un elemento o todos los elementos de una secuencia son verdaderos.
- $str()$: Convierte un objeto en una cadena de texto.
- $list()$: Convierte un objeto en una lista.
- $dict()$: Crea un diccionario a partir de una secuencia de pares clave-valor.

1.1.9.1 Módulos y Funciones Personalizadas. Puedes organizar tus funciones en módulos para mantener tu código limpio y estructurado. Los módulos son archivos que contienen funciones y variables que pueden importarse en otros programas. Para utilizar funciones de un módulo, debes importarlo.

1.1.10. Programación orientada a objetos (POO)

La Programación Orientada a Objetos (POO) es un enfoque de programación que se fundamenta en la noción de "objetos". Estos objetos, representativos de instancias de clases, sirven como modelos o prototipos que especifican las características y comportamientos compartidos por un conjunto de objetos.

En la POO, los objetos son entidades que encapsulan datos y funciones relacionadas. Cada objeto se deriva de una clase, siendo esta última una plantilla que define propiedades y

métodos compartidos por todos los objetos de esa categoría. Entre los conceptos clave de la POO se encuentran:

- Clases: Definen las características compartidas de un grupo de objetos, actuando como planos para la creación de objetos individuales.
- Objetos: Representan instancias concretas de clases y encarnan entidades del mundo real, con atributos (datos) y métodos (funciones) asociados.
- Encapsulamiento: Consiste en ocultar los detalles internos de un objeto y revelar únicamente lo necesario para interactuar con él. Esto facilita la modularización y organización del código.
- Herencia: Permite a una clase heredar atributos y métodos de otra, fomentando la reutilización de código y la creación de jerarquías de clases.
- Polimorfismo: Permite que un objeto exhiba comportamientos distintos según el contexto, donde un mismo método puede tener implementaciones diferentes en clases diversas.

La POO se emplea para modelar y resolver problemas de forma más intuitiva y estructurada, proporcionando una vía eficiente para organizar el código, favoreciendo la reutilización y el mantenimiento, y reflejando de manera más precisa la estructura y relaciones del mundo real.

1.1.11. Programación orientada a objetos en Python

1.1.11.1 Clases y Objetos. Las clases son componentes esenciales de la implementación de la Programación Orientada a Objetos (POO) de Python. Una clase sirve como modelo o plantilla que establece la composición y características de los objetos que se derivan de ella. En esencia, una clase contiene los comportamientos (llamados métodos) y

los datos (llamados atributos) que definen esos objetos. Esto facilita la escritura de código modular y estructurado, lo que permite representar elementos del mundo real de una forma más natural.

Python tiene una sintaxis fácil de usar y adaptable para crear clases. Una clase con propiedades como modelo, color y año, así como métodos que describen su funcionamiento, como arrancar, detenerse y cambiar de velocidad, puede definirse, por ejemplo, para representar un concepto como "coche". Cuando se crea un objeto específico a partir de esta clase, se crea una instancia única que hereda los rasgos y comportamientos especificados en la clase, pero con valores específicos para sus atributos.

La relación entre clases y objetos en Python es esencial para comprender la POO, o programación orientada a objetos. Un objeto es una creación particular que se basa en una clase, la cual puede considerarse como un plano para su construcción. Se pueden crear sistemas complejos y dinámicos debido a que los objetos son entidades activas que pueden comunicarse entre sí a través de métodos especificados en sus clases respectivas.

Un gran beneficio que ofrecen las clases y objetos en Python es la reutilización de código. Una clase puede ser instanciada más de una vez para producir objetos distintos con propiedades y funciones idénticas una vez que ha sido definida. Esto reduce la duplicación de código y permite modificar de manera centralizada una clase para impactar a todos sus objetos, lo que mejora la eficiencia en el desarrollo de software.

1.1.11.2 Constructor. Los constructores son métodos únicos en las clases de Python que se utilizan para inicializar objetos al crear una instancia de esa clase. Establecer el estado inicial de un objeto asignando valores a sus propiedades es la función principal del constructor principal de Python, __init__(). En la Programación Orientada a Objetos, este método es crucial porque permite que los objetos de la clase se configuren de manera

consistente desde el principio.

En Python, el nombre __init__() es una convención altamente importante: este método se ejecuta automáticamente sin necesidad de ser llamado explícitamente cada vez que se crea un nuevo objeto de una clase. El constructor define los parámetros necesarios para construir un objeto; el argumento especial self se utiliza para asignar estos valores a los atributos de la instancia.

La capacidad del constructor para admitir argumentos es una de sus características más sólidas, ya que ofrece una gran libertad al construir cosas. Al inicializar los atributos del objeto con valores específicos utilizando estos parámetros suministrados al constructor, se hace posible la personalización en el momento de la creación de la instancia.

El constructor, que es el primer método llamado al construir un objeto, es esencial para establecer el comportamiento inicial del objeto. Además de inicializar atributos, el constructor puede realizar tareas adicionales necesarias para que el objeto funcione correctamente, como llamar a otros métodos de la clase o inicializar configuraciones o validaciones de datos.

1.1.11.3 Atributos. Los atributos en Python son bloques fundamentales para la programación orientada a objetos. Estas variables, asociadas a una clase, otorgan a cada instancia de esa clase rasgos o información particulares.

En Python, los atributos pueden clasificarse como instancias o basados en clases. Los objetos producidos a partir de clases tienen atributos únicos llamados atributos de instancia. El método constructor __init__() define estas características, que pueden ser accedidas mediante notación de punto. Nombre, edad y género son ejemplos de características de instancia de la clase Persona.

Por otro lado, los atributos de clase son compartidos por todas las instancias de una clase y se definen explícitamente dentro de la clase, independientemente de cualquier metodología. Estos atributos se acceden a través de la misma clase. "Nacionalidad", por ejemplo, podría ser una característica de clase en la clase Persona.

Python permite la manipulación dinámica de atributos, lo que posibilita la creación, modificación y eliminación de atributos mientras se lleva a cabo una operación. Esto hace que la programación sea más flexible, ya que los atributos de los objetos pueden modificarse según las necesidades del programa.

Además, las convenciones de nombres se pueden utilizar para regular la visibilidad de los atributos en diferentes niveles. Los atributos privados solo pueden ser accedidos al agregar un doble guion bajo al nombre, los atributos protegidos solo pueden ser accedidos siguiendo una convención de nombres especificada, y los atributos públicos pueden ser accedidos desde fuera de la clase. Esta convención ayuda a controlar y preservar la integridad de los datos.

1.1.11.4 Métodos. Las funciones definidas dentro de una clase se llaman métodos, y se utilizan para realizar tareas específicas en objetos derivados de esa clase. Estas técnicas pueden cambiar el estado interno de un objeto o proporcionar funcionalidades que afectan cómo se comporta el elemento.

Los métodos pueden ser métodos de instancia. Estos métodos trabajan en instancias específicas de la clase y toman una referencia al objeto como su primer argumento normalmente llamado "self" por defecto. Estos métodos pueden ser invocados a través de la instancia misma y permiten interactuar con sus características.

Por otra parte, los métodos de clase utilizan atributos de clase en lugar de propiedades de instancia y se especifican usando el decorador @classmethod. En lugar de una sola

instancia, aceptan la propia clase como su primer parámetro (generalmente designado como “cls” por convención). Estos métodos pueden aplicarse a tareas que involucran a la clase en sí misma, como administrar variables de clase o generar instancias alternativas.

Aparte de esto, existen métodos estáticos que se definen utilizando el decorador @staticmethod, que no obtienen una referencia a la instancia o clase automáticamente. Aunque estos métodos se incluyen en el ámbito de la clase con fines organizativos y de diseño, operan más como funciones estándar.

Mientras que todos los métodos en Python son accesibles en tiempo de ejecución, también pueden tener diferentes grados de accesibilidad, al igual que los atributos, mediante el empleo de convenciones de nomenclatura para los métodos públicos, protegidos y privados.

1.1.11.5 Encapsulamiento. La encapsulación en Python es una técnica de programación orientada a objetos que se centra en ocultar la implementación central de una clase y limitar el acceso directo a determinados métodos y atributos. Al restringir el acceso a los datos y limitar su modificación a los métodos designados de la clase, esta idea pretende mejorar la seguridad y la integridad de los datos.

En Python, no existen modificadores de acceso como en otros lenguajes, por ejemplo “public” “private” o “protected”, pero se sigue el principio de “convención de nombre” para indicar la visibilidad de los atributos y métodos. Por convención, los atributos y métodos que comienzan con un guion bajo como _atributo o _metodo() se consideran como "privados" y no se deberían acceder desde fuera de la clase.

Dado que los atributos y funciones “privados” aún pueden ser accedidos desde fuera de la clase, la convención de privacidad de Python depende más del consenso de los

desarrolladores que de una limitación estricta. Puedes utilizar ciertas convenciones de nomenclatura para acceder a estos miembros, aunque no es aconsejable.

1.1.11.6 Herencia. Una idea clave en la programación orientada a objetos en Python es la herencia, que permite a una clase heredar propiedades y funciones de otra. Esto implica que una clase, denominada clase padre o superclase, puede utilizar y ampliar las capacidades de otra clase, denominada clase hija o subclase.La sintaxis de declaración de clase se utiliza para crear una relación de herencia; el nombre de la clase hija va seguido de la clase padre entre paréntesis. La clase Hijo (Padre): es un ejemplo. La clase hija puede añadir nuevos métodos, modificar los existentes y definir sus propios atributos. También hereda todos los métodos y atributos de la clase padre.

El mecanismo de herencia de Python permite crear una jerarquía de clases, lo que fomenta la disposición lógica de las clases y facilita la reutilización del código. Esto conduce a una arquitectura de código más modular y fácil de mantener, ya que las clases más especializadas (los hijos) pueden heredar rasgos y comportamientos de clases más genéricas (los padres).

1.1.11.7 Polimorfismo. Uno de los conceptos clave de la programación orientada a objetos en Python es el polimorfismo, que permite tratar de forma coherente objetos de clases diferentes. Esta característica se basa en la capacidad de un objeto para adoptar diversas formas y mostrar diversos comportamientos en función de la situación en la que se encuentre.

El polimorfismo de sobrecarga de operadores y el polimorfismo de herencia son las dos formas principales en que se expresa el polimorfismo en Python. En el primer caso, los operadores pueden mostrar distintos comportamientos según el tipo de operandos que utilicen. El operador “+”, por ejemplo, puede concatenar textos o hacer sumas aritméticas entre enteros.

La base del polimorfismo basado en la herencia es el uso de métodos con el mismo nombre, pero con acciones específicas de clase en varias clases. Esto hace posible que un método responda de forma diferente a la clase de objeto que lo invoca. Por ejemplo, un método make_sound() en la clase Perro también podría estar en la clase Gato, pero cuando es invocado, los resultados serán diferentes.

En Python, el polimorfismo también se refiere al uso de funciones y métodos que pueden trabajar de forma fiable con argumentos de varios tipos. Como ilustración, considere una función que recibe varios tipos de objetos y, dependiendo del tipo que obtenga, ejecuta acciones particulares.

El polimorfismo de Python facilita la escritura de código más versátil y adaptable al permitir una interacción consistente con objetos de varias clases. Como resultado, los diseños de código se vuelven más claros, modulares y manejables, ya que las estructuras genéricas pueden manejar una variedad de tipos de objetos sin necesidad de manejar las especificidades de cada clase por separado.

1.2. Metodología de gestión de proyectos Scrum

El éxito en el entorno empresarial actual depende en gran medida de la eficacia de la gestión de proyectos, dada su naturaleza siempre cambiante y ferozmente competitiva. Dentro de la familia de técnicas ágiles, la metodología Scrum se ha dado a conocer como una técnica potente y flexible que ha cambiado por completo la forma en que los equipos de diversos sectores abordan la gestión de proyectos y la creación de productos.

Es especialmente eficaz en entornos dinámicos, donde los requisitos del proyecto pueden cambiar o no estar completamente definidos inicialmente. A continuación, se presenta una exhaustiva descripción de la metodología Scrum, destacando sus principios fundamentales, roles, artefactos y eventos clave.

1.2.1. Principios Básicos

La metodología Scrum fue desarrollada por primera vez en la década de 1980 por Jeff Sutherland y Ken Schwaber como marco ágil de gestión de proyectos. Por aquel entonces, estaban influidos por las técnicas de gestión de la producción de Toyota.

Los fundamentos de Scrum se basan en el concepto de gestión ágil, que hace hincapié en la adaptabilidad, la flexibilidad y la entrega continua de valor al cliente. Tres pilares clave forman su base: inspección, adaptabilidad y transparencia. La inspección es la evaluación continua del trabajo que se está realizando, la adaptación es el proceso de modificar y mejorar continuamente los resultados, y la transparencia es la capacidad de ver todas las facetas del proceso de trabajo.

Scrum se distingue por su metodología incremental e iterativa, en la que las tareas se dividen en breves intervalos conocidos como sprints. Cada sprint, que dura de dos a cuatro semanas por término medio, tiene objetivos específicos y genera entregables útiles. Con esta estrategia, los equipos pueden reaccionar con rapidez a las modificaciones y obtener aportaciones continuas, lo que eleva la calidad del producto final y aumenta la satisfacción del cliente.

1.2.2. Roles en Scrum

Los roles de Scrum son esenciales para el uso eficiente de esta técnica ágil de gestión de proyectos. Cada uno de estos puestos de trabajo está bien definido, con deberes particulares que promueven tanto el éxito del equipo y del proyecto en general.

Uno de los roles más importantes en Scrum es el Product Owner. Esta persona tiene la responsabilidad de abogar en nombre del cliente o de la empresa. Su trabajo principal es optimizar el valor del producto que el equipo Scrum desarrolla. Establecer y priorizar los requisitos del producto en estrecha colaboración con el equipo, el Product Owner mantiene

la visión global en mente mientras decide qué características implementar primero y en qué secuencia.

Otra posición crucial es la del Scrum Master. Su principal deber es ayudar al proceso Scrum y garantizar que se utiliza correctamente. Como líder servidor, el Scrum Master ayuda al equipo mediante la eliminación de obstáculos, fomentando el trabajo en equipo, y protegiéndolos de las distracciones externas. También facilita las reuniones y actividades de Scrum, apoya la mejora continua, y ayuda al equipo en la comprensión y aplicación de conceptos ágiles.

La tercera posición en Scrum es el equipo de desarrollo. Al final de cada sprint, este equipo de profesionales autoorganizados y multifuncionales se encarga de convertir los elementos del backlog del producto en incrementos de trabajo posiblemente entregables. El equipo de desarrollo trabaja en estrecha colaboración con el Scrum Master para seguir los métodos y principios de Scrum, y con el Producto Owner para entender las necesidades.

1.2.3. Artefactos en Scrum:

Los artefactos son componentes esenciales de Scrum que ofrecen visibilidad y transparencia en el trabajo completado y sin terminar en un proyecto. El Backlog del Producto, el Backlog del Sprint, y el Incremento son los tres tipos de artefactos.

Una lista dinámica y priorizada de todas las características, especificaciones, correcciones inacabadas y mejoras de un producto se denomina cartera de productos. Siempre está cambiando, y el Propietario del Producto se encarga de gestionarla, establecer prioridades y actualizarla. Con el tiempo, este artefacto establece el alcance del proyecto y representa las necesidades del cliente.

Una colección de tareas elegidas del Product Backlog para ser trabajadas en un sprint en particular se llama Sprint Backlog. La carga de trabajo del equipo de desarrollo durante

ese tiempo está representada por estas tareas. El equipo de desarrollo, que determina cómo se llevarán a cabo esas tareas, es el encargado de crear el Sprint Backlog, que se establece durante la reunión de planificación del sprint.

El número total de elementos del Product Backlog que se han terminado en un sprint se conoce como Incremento. Es un paso tangible hacia el producto final y debe ser enviable, lo que significa que debe ser de un calibre que permita su distribución, si así se determina. El objetivo es tener un incremento de trabajo que mejore el producto al final de cada sprint.

Para que el equipo, el propietario del producto y otras partes interesadas comprendan el progreso del proyecto, estos artefactos Scrum son herramientas esenciales. Ayudan a mantener un enfoque claro en los objetivos del producto, promueven la colaboración y son transparentes sobre el trabajo que se está realizando. La entrega regular y continua de valor al cliente depende de su gestión y actualización continuas.

1.2.4. Beneficios de Scrum

Scrum es un popular estilo de gestión de proyectos que atrae a un amplio abanico de sectores debido a sus numerosas y notables ventajas. En primer lugar, fomenta la adaptabilidad y la flexibilidad al permitir una reacción rápida a las modificaciones de las especificaciones del proyecto. Scrum permite la integración de aportaciones y ajustes continuos dividiendo el trabajo en breves iteraciones, o sprints. Este enfoque genera productos que se adaptan mejor a la evolución del mercado o a las demandas de los consumidores.

La transparencia es otro factor importante. A través de herramientas como el Product Backlog y el Incremento, Scrum da a las partes interesadas acceso al progreso del proyecto para que puedan evaluar el estado del trabajo y tomar decisiones acertadas. El equipo y las

partes interesadas del proyecto trabajan juntos de manera más eficaz y se comunican más eficazmente cuando hay transparencia.

Una característica que distingue a Scrum es la entrega gradual y continua de valor. El enfoque pone un gran énfasis en la creación de incrementos potencialmente entregables al final de cada sprint, lo que permite a los usuarios acceder a características utilizables con mayor regularidad. Como resultado, aumenta la satisfacción del cliente y mejora la flexibilidad del mercado.

Scrum también fomenta la responsabilidad y la autoorganización en los equipos. Dar al equipo de desarrollo la libertad de organizar y llevar a cabo su trabajo fomenta la motivación y la innovación, lo que con frecuencia conduce a mejores productos y equipos más comprometidos y felices.

Por no hablar de que el proceso de la metodología Scrum está impregnado de una mentalidad de mejora continua. Los equipos identifican áreas de mejora y establecen medidas correctivas para maximizar el rendimiento y la eficiencia en el siguiente ciclo de trabajo a través de reuniones retrospectivas celebradas al final de cada sprint.

1.3. Caracterización guía “Guía para el cálculo y la selección de apoyos en estructuras de media y baja tensión para sistemas de distribución actualizado a normatividad vigente, RETIE y NTC2050”

En esta sección, detallaremos y analizaremos la guía dedicada a cálculos mecánicos y selección de estructuras, que sirve como base para desarrollar nuestro software.

El propósito central de esta guía es proporcionar una guía exhaustiva sobre cómo realizar los cálculos necesarios y la selección adecuada de estructuras destinadas a baja y media tensión.

Dirigida específicamente a estudiantes con conocimientos en el área eléctrica, la guía se ha estructurado lógica y fácil de entender. El contenido se organiza meticulosamente, iniciando con la identificación de la altura de la estructura correspondiente al nivel de tensión adecuado. Seguidamente, se presenta una tabla detallada que ofrece información crucial sobre las estructuras disponibles, categorizadas por su altura, carga de rotura, diámetro de la cima y diámetro de la base.

La guía se desglosa en cuatro etapas esenciales:

a) Fuerzas que actúan en condición normal.

- Fuerza por presión del viento: La estructura, los conductores y los aisladores experimentan una fuerza lateral generada por la acción del viento. Esta fuerza varía según la forma de los elementos y la ubicación geográfica de la estructura. Por consiguiente, es necesario calcular una fuerza específica por presión del viento para cada tipo de elemento presente en la estructura. La fórmula para calcular esta fuerza (Fv) se expresa como $Fv = Q\ x\ Kz\ x\ Kzt\ x\ V^2\ x\ G\ x\ Cf\ x\ A$, donde Q, Kz, Kzt, V, G, Cf y A son parámetros que consideran la magnitud y características del viento, la geometría de la estructura, la ubicación y otros factores relevantes.
- Fuerza por desequilibrio de tensiones mecánicas: Se presta atención a la carga generada por la tensión mecánica de los conductores a lo largo de su extensión, sobre todo cuando las tensiones en tramos cercanos no son iguales. Esto es especialmente relevante en estructuras con desniveles, en los extremos y en los puntos de anclaje. La fórmula asociada a esta consideración es

$FTW = |TH1 - TH2|$, donde FTW representa la carga por tensión mecánica y TH1 y TH2 son las tensiones respectivas en los vanos adyacentes.

- Fuerza por cambio de dirección de la línea: Cuando una línea cambia de dirección, se produce una fuerza transversal como consecuencia de la combinación de tensiones en los tramos cercanos. Esto ocurre debido a la variación en la dirección de la línea y resulta en una fuerza lateral que es el resultado de la suma de las tensiones en los vanos adyacentes, $FT\theta = (TH1 + TH2)\sin\frac{\theta}{2}$. Cuando la línea experimenta un cambio de dirección, la fuerza originada por el desequilibrio de tensiones mecánicas se incrementa al ser multiplicada por un factor coseno $FTW = (TH1 - TH2)\cos\frac{\theta}{2}$, donde TH es la tensión mecánica de los vanos adyacentes y θ el ángulo de desviación.

b) Fuerzas que actúan en condición anormal.

- Fuerza por conductor roto: Se considera la posibilidad de un conductor roto en una fase, mientras que las demás fases y el cable de guarda están intactos, o en el caso contrario, donde el cable de guarda está roto pero las fases se encuentran en buen estado. Esta situación se evalúa en condiciones climáticas de viento reducido y temperaturas coincidentes. En este escenario, la tensión máxima sobre la estructura es generada por el cable íntegro del vano adyacente, la ecuación es $FWR = |TWR|$, donde TWR es la tensión mecánica del conductor adyacente.
- Fuerza por desequilibrio del 50% de tensiones mecánicas: Se tiene en cuenta un desequilibrio del 50% en las tensiones de todos los cables en la estructura cuando se enfrenta a condiciones de viento máximo y temperatura

coincidente., $FT50\% = 0.5\,x\,|THVM|$, donde THVM es la mayor tensión mecánica del vano adyacente.

c) Momento de fuerzas sobre la estructura.

- Momento generado por fuerzas: Se toma como punto de partida la idea de que los momentos actúan a la altura efectiva, y esto se calcula durante el análisis de las fuerzas generadas por la presión del viento. Para los aisladores y conductores, se utiliza la altura promedio de estos elementos, mientras que para las estructuras se considera el centroide. En casos donde hay más de un conductor o aislador, simplemente multiplicamos la cantidad presente para tener en cuenta cada uno de ellos. $M = N\,x\,F\,x\,Z$.
- Momento resistente: De acuerdo con la normativa NTC 1329, el momento resistente se genera a partir de la carga máxima de rotura ubicada a una distancia de 20 centímetros desde la parte superior de la estructura. $MR = CR\,x\,(L - Le - 0.2)$, donde CR es la carga de rotura, L la longitud de la estructura y Le la longitud de empotramiento.

d) Verificación de los postes.

- Momento actuante en condición normal: En situaciones normales, el momento que realmente importa es la suma de todos los momentos, ya sea en dirección longitudinal o transversal, que se encuentran en la estructura. Sin embargo, no se considera el momento resistente en este cálculo. $MA = \sqrt{(MTw)^2 + (MVs + MVa + MVw + MT\theta)^2}$, donde MTW es momento por desequilibrio de tensiones mecánicas, MVw, MVa, MVs es

momento por presión del viento en conductores, aisladores y estructura, respectivamente y MTθ es momento por cambio de dirección de la línea.

- Momento actuante en condición anormal: En situaciones inusuales, como cuando hay un conductor roto y un desequilibrio del 50% en las tensiones mecánicas, el momento actuante es únicamente el momento estudiado en la hipótesis. $MA1 = MWR, MA2 = MT50\%$.
- Verificación final: Se compara el momento que se presenta en situaciones normales y anormales con el momento resistente de la estructura, considerando un factor de seguridad de 2.5, que es aplicable a postes de concreto. $\frac{MR}{MA} > 2.5$, donde MR es momento resistente y MA momento actuante.

La guía proporciona detalladamente ecuaciones y consideraciones cruciales para la selección apropiada de cada variable. Al final del proceso, se lleva a cabo una verificación final para asegurar la elección correcta de la estructura.

Además, la guía se presenta de manera digital y sin costo alguno, priorizando la accesibilidad. El lenguaje empleado es de fácil comprensión, inclusive para aquellos con conocimientos técnicos limitados. Se incorporan gráficos y diagramas para mejorar la claridad, y se incluye un ejercicio práctico explicado paso a paso para facilitar aún más la comprensión y aplicación de los conceptos descritos en la guía.

2. Desarrollo del proyecto

El objetivo principal es ofrecer a los estudiantes y profesionales de la ingeniería eléctrica una solución innovadora tanto como accesible, agilizando y simplificando en gran medida los procedimientos de diseño de sistemas eléctricos de media tensión. Se eligió

Python como lenguaje de programación para lograr este objetivo debido a su popularidad, facilidad de uso y rico ecosistema de bibliotecas.

Por otra parte, para la aplicación de la metodología Scrum en el desarrollo de nuestra tesis hemos seguido un enfoque ágil que ha demostrado ser efectivo para gestionar la complejidad del proyecto y garantizar entregas incrementales de alta calidad. Durante el periodo de tres meses, hemos dividido el trabajo en sprints, cada uno con objetivos específicos. En el primer sprint, se llevó a cabo la planificación del proyecto y la definición del backlog del producto, seguido por el desarrollo inicial de la interfaz para ingresar datos del proyecto. En el segundo sprint, nos enfocamos en la revisión del trabajo anterior y planificamos el siguiente sprint, durante el cual implementamos la funcionalidad para el cálculo de postes. Finalmente, en el tercer sprint, concluimos el desarrollo del software y llevamos a cabo pruebas integrales.

A lo largo de todo el proceso, hemos incorporado eventos recurrentes como las reuniones diarias de sincronización (Daily Stand-ups), revisiones al final de cada sprint (Sprint Review) para evaluar y demostrar el trabajo realizado, y retrospectivas al final de cada sprint (Sprint Retrospective) para reflexionar sobre el proceso y realizar mejoras. Este enfoque iterativo y colaborativo ha facilitado la adaptación continua a medida que se obtienen nuevos conocimientos y se ajustan las prioridades del proyecto, asegurando así una implementación exitosa de las funcionalidades clave en el contexto de nuestra tesis.

2.1. Componentes de la solución

Nuestro enfoque se basa en la creación de elementos esenciales, que van desde la administración eficaz de los procesos internos hasta la estructura del diseño. Entre las características importantes figuran la adaptabilidad del sistema y el correcto funcionamiento

de la interfaz. Cada una de las funciones mencionadas es esencial para la creación del producto final, que está hecho para satisfacer las demandas del usuario final.

2.1.1. Tipos de usuario

Este proyecto está dirigido a varios tipos de usuarios, cada uno con demandas y objetivos distintos en los campos de la ingeniería eléctrica y el diseño de sistemas de media tensión. Se ha llegado a la conclusión de que el proyecto puede dirigirse a los siguientes tipos de usuarios:

- **Estudiantes de Ingeniería Eléctrica:** teniendo en cuenta que el mercado cuenta con pocas o nulas alternativas a la comprobación de los cálculos asociadas a las estructuras de media tensión, se ha dado como prioridad en la realización del software el poder comprobar de manera eficaz los cálculos realizados teóricamente por parte del estudiante, permitiendo así contrastar la información obtenida, lo cual permite el fortalecimiento de su formación académica.
- **Ingenieros eléctricos:** ingenieros que se inician en el campo del diseño y la planificación de sistemas eléctricos de media tensión. Estos jóvenes profesionales deben realizar cálculos mecánicos para garantizar la eficacia y seguridad de las estructuras, pero a menudo se topan con el gasto que supone un software especializado.
- **Pequeñas empresas:** empresas con recursos financieros limitados que buscan una solución accesible y eficiente para realizar cálculos mecánicos en sus proyectos de media tensión.

- **Colaboradores en el desarrollo:** desarrolladores y programadores que quieren ampliar el código fuente del software, mejorarlo y modificarlo para adaptarlo a las demandas cambiantes de la base de consumidores.

2.1.2. Algoritmos y funcionalidades

Un algoritmo es un conjunto de pasos o reglas precisas y bien definidas que especifican cómo llevar a cabo un cálculo o resolver un problema concreto. Dicho de otro modo, es una colección detallada de pautas o directrices que, cuando se cumplen, dan como resultado la resolución de un problema o la realización de una tarea determinada.

Estas pautas se encuentran plasmadas en la Guía para el cálculo y la selección de apoyos en estructuras de media y baja tensión para sistemas de distribución actualizado a normatividad vigente, RETIE y NTC2050. A continuación, se detallan algunos de los algoritmos clave y sus funcionalidades asociadas:

1. **Fuerzas que actúan en condición normal:** En circunstancias típicas, se integran algoritmos particulares para evaluar la carga de presión del viento, calcular las fuerzas resultantes de desequilibrios en las tensiones mecánicas y examinar el impacto de las variaciones de dirección en la línea eléctrica. Estos algoritmos ofrecen una evaluación exhaustiva de las fuerzas en juego mientras la estructura funciona normalmente.
2. **Fuerzas que actúan en condición anormal:** El software tiene en cuenta las fuerzas resultantes de la rotura de conductores y los desequilibrios especiales del 50% en las tensiones mecánicas en circunstancias excepcionales. Esto garantiza una reacción completa ante circunstancias imprevistas, mejorando la resistencia y la seguridad de la estructura en casos atípicos.
3. **Momentos de fuerza sobre las estructuras:** Se utilizan algoritmos especializados para determinar los momentos de fuerza que actúan sobre las estructuras. Estos

algoritmos tienen en cuenta los momentos de fuerza producidos por diferentes fuerzas, desequilibrios mecánicos de tensión, cambios de dirección de las líneas eléctricas, rotura de conductores y ciertos desajustes del 50% en las tensiones. Además, se determinan los momentos resistivos intrínsecos de la estructura, facilitando así la evaluación de su capacidad para soportar momentos externos y garantizando la estabilidad estructural en diversas condiciones.

4. **Verificación de Postes de Concreto:** El software realiza cálculos precisos del momento de acción tanto en circunstancias normales como excepcionales como parte del procedimiento de inspección de postes de hormigón. Mediante un enfoque meticuloso, es posible evaluar la capacidad de resistencia del poste y asegurarse de que cumple las normas de seguridad y las reglas vigentes.
5. **Interfaz de Usuario:** Al ofrecer herramientas prácticas y fáciles de usar, la interfaz de usuario se ha creado para maximizar la experiencia del usuario. Aunque actualmente no permite compartir proyectos o resultados directamente entre usuarios, su principal objetivo es agilizar la introducción de datos y producir resultados legibles.

2.1.3. Planificación de requerimientos

Durante esta etapa, mediante reuniones con el director y el codirector se discutió el alcance del proyecto además de examinar varios problemas relacionados con la aplicación de los algoritmos. Con el fin de presentar una visión general de la arquitectura del software y trazar el plan de acción para las siguientes fases de desarrollo. Por lo tanto, en esta sección se detallarán las necesidades funcionales y no funcionales del programa, así como las características necesarias para garantizar su eficacia, usabilidad y fiabilidad.

2.1.3.1 Requerimientos funcionales. Los requisitos funcionales se centran en las

tareas concretas que debe realizar el software para cumplir sus objetivos. Para que los usuarios tengan una experiencia fluida y eficaz, estos elementos operativos y de rendimiento del sistema son cruciales para cumplir con lo anteriormente mencionado. Los aspectos funcionales obtenidos después del análisis detallado para los usuarios son:

2.1.3.2 Funcionalidades Básicas. En el marco del desarrollo de este software especializado en cálculos mecánicos para estructuras de concreto de media tensión, es crucial comprender las funciones fundamentales que constituirán la esencia operativa del sistema. Nos enfocaremos en los requerimientos funcionales que definen las capacidades operativas y de rendimiento del software.

1. **Ingreso de datos:** El usuario puede facilitar detalles cruciales como las condiciones de carga y las especificaciones estructurales, nombre del proyecto, ubicación del proyecto. Esta etapa incluye la validación de datos, que verifica la precisión y coherencia de los datos introducidos, además de servir como punto de partida crucial para el análisis. La calidad y precisión de estos datos iniciales afectará directamente a la eficacia de los cálculos mecánicos posteriores, consolidando la base de funcionamiento del software y demostrando su fiabilidad como herramienta de ingeniería eléctrica.
2. **Tratamiento de los datos:** Una vez que el usuario ha ingresado los datos relevantes a través de la interfaz de usuario, el software implementa algoritmos específicos para llevar a cabo el procesamiento y tratamiento de estos datos. Este proceso es esencial para realizar cálculos mecánicos precisos y obtener resultados confiables.

   Los algoritmos utilizan los parámetros interpretados para realizar cálculos mecánicos. Esto implica calcular fuerzas y momentos tanto en circunstancias típicas como inusuales, teniendo en cuenta variables como la presión del viento,

desequilibrios en las tensiones mecánicas, cambios en la dirección de la línea, rotura de conductores y otros sucesos cruciales.

3. **Visualización de los resultados:** Una característica crucial del software es la visualización de los resultados, que ofrece a los usuarios una comprensión fácil de los datos producidos por los cálculos mecánicos. Este aspecto funcional pretende ayudar al usuario en la interpretación y la toma de decisiones, además de presentar eficazmente la información. La interfaz presenta un resumen claro de los parámetros clave, destacando las fuerzas y momentos más relevantes. Esto ayuda a los usuarios a identificar rápidamente los aspectos críticos del diseño y tomar decisiones informadas.

2.1.3.3 Requerimientos No Funcionales. Los requisitos no funcionales abarcan partes del software que no están directamente relacionadas con sus funciones específicas, sino con sus características generales, limitaciones y prestaciones que afectan a la eficacia del sistema y a la experiencia del usuario. A continuación, se detallan algunos requerimientos no funcionales clave para el software:

1. **Desempeño:** La capacidad de un programa informático para llevar a cabo actividades de forma eficaz y eficiente, cumpliendo unos requisitos de rendimiento predeterminados, se conoce como rendimiento del programa informático. La importancia del rendimiento del software varía en función del tipo de aplicación y puede evaluarse de varias maneras.
2. **Usabilidad:** Para garantizar que los usuarios puedan introducir datos, comprender los resultados y realizar ajustes de forma rápida y sencilla, la interfaz de usuario debe ser fácil de usar e intuitiva.

3. **Compatibilidad:** En la creación y distribución del software se utilizará un archivo ejecutable para proporcionar un acceso sencillo y universal desde cualquier PC. El objetivo de este método es ofrecer a los usuarios una solución que pueda desplegarse con facilidad, eliminando la necesidad de configuraciones intrincadas o requisitos técnicos previos sustanciales.
4. **Normativas y Estándares:** Cumplir con las normativas y estándares aplicables en el diseño de estructuras de media tensión, garantizando que el software se ajuste a las regulaciones vigentes.

2.2. Diseño de prototipo

Una vez que se abordaron los requisitos, se procedió a la elaboración de casos de uso, diagramas de actividades, diagramas de clases y la arquitectura de software. Todo lo mencionado anteriormente establece los cimientos para la creación del prototipo. A continuación, se describen las acciones clave realizadas durante el diseño, las cuales son esenciales para evaluar la solución en términos de implementación, considerando únicamente la perspectiva de las ciencias de la computación.

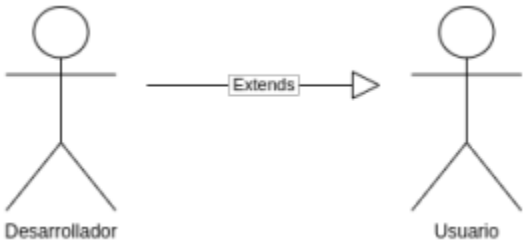
1. Registrar características del proyecto.
2. Registrar características de la zona.
3. Registrar datos de conductores y aisladores.
4. Inicio del algoritmo.
5. Fin del algoritmo.
6. Mostrar resultados.

Asimismo, en la fase de diseño del prototipo se incorporaron diagramas de actores con el fin de reconocer a los usuarios principales que participan en el sistema. El rol 'Desarrollador' se deriva del rol 'Usuario', lo que implica que el rol 'Desarrollador' constituye

una versión más avanzada del rol 'Usuario', al compartir las mismas funcionalidades y añadir otras nuevas.

Figura 2

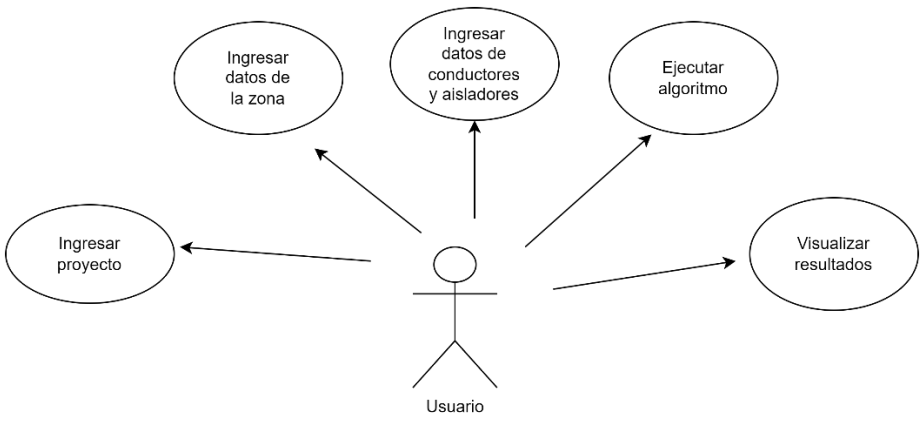
Diagrama de actores del sistema



Una vez que se han reconocido tanto los actores como los requisitos funcionales, se ha sido capaz de elaborar los correspondientes diagramas de casos de uso. La figura 2 exhiben el diagrama de casos de uso con el propósito de representar de manera simplificada y modularizada las funciones del software.

Figura 3

Diagrama de casos de uso



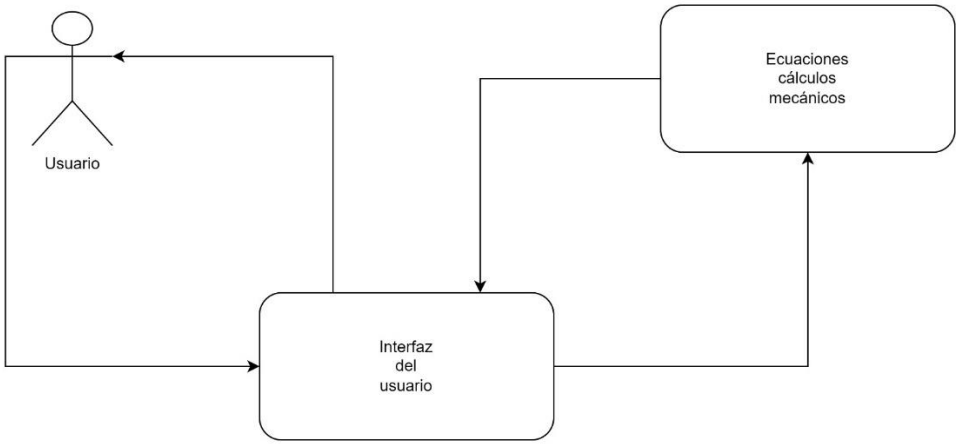
El componente central es el 'Usuario', quien desempeña un papel fundamental como el actor principal del software, consumiendo los productos y/o servicios disponibles. El 'Usuario' tiene la capacidad de generar y observar diversas ejecuciones de su propia autoría.

2.3. Arquitectura propuesta

La estructura conceptual de esta investigación se presenta de manera gráfica en la figura 3. En dicha representación, se destacan dos fases fundamentales que conforman la arquitectura propuesta: la interfaz del usuario y el algoritmo encargado de gestionar todas las operaciones relacionadas con las ecuaciones esenciales para el cálculo mecánico de los postes de concreto.

Figura 4

Representación estructura conceptual



- **Interfaz del usuario:** La interfaz de usuario de nuestro software se creó principalmente empleando la biblioteca *tkinter* de Python, una herramienta versátil y sólida especializada en la elaboración de interfaces gráficas. *tkinter* presenta una variedad extensa de widgets que simplifican la interacción del usuario con la aplicación, asegurando al mismo tiempo una experiencia atractiva e intuitiva. Debido a su capacidad de ser utilizada en múltiples plataformas, nuestra interfaz es

compatible con diversos sistemas operativos, garantizando la accesibilidad y la usabilidad del software en distintos contextos. La selección de *tkinter* se basa en su integración fluida con Python, lo que nos ha permitido desarrollar una interfaz eficaz y funcional que complementa de manera efectiva el poderoso algoritmo subyacente de cálculos mecánicos. Igualmente se utilizaron librerías adicionales como *Cerberus* la cual nos ayudó para la validación de los datos y *pillow* para la implementación de imágenes en la interfaz.

- **Algoritmo de cálculos mecánicos:** En cuanto al algoritmo, podríamos decir que es el cerebro detrás de todo. Aquí es donde guardamos todas las ecuaciones y la lógica necesaria para realizar los cálculos mecánicos específicos que se aplican a los postes de concreto. La solidez y eficiencia de este algoritmo son cruciales para asegurar que los resultados sean precisos y confiables, brindando así un respaldo técnico sólido para las decisiones en diseño e ingeniería estructural.

  El algoritmo para realizar los cálculos mecánicos en nuestra aplicación fue desarrollado mediante el uso del lenguaje de programación Python, aplicando una metodología orientada a objetos. Esta elección permitió organizar el código de manera modular, facilitando su mantenimiento y posibilitando futuras extensiones del sistema. Se utilizaron librerías como *numpy* para el manejo de las ecuaciones y *panda* para estructurar, manipular y operar datos como matrices.

  Adicionalmente, se implementó el formato de *Atomic Design* en la estructura del algoritmo. Este enfoque descompone la interfaz en unidades fundamentales denominadas "átomos", que son combinadas para formar componentes más complejos. En el caso del algoritmo, esta metodología nos proporcionó una estructura

modular y escalable, donde cada "átomo" representa una funcionalidad específica y se integra eficientemente para construir operaciones más avanzadas.

La combinación de la programación orientada a objetos y la metodología *Atomic Design* se integra de manera sinérgica, resultando en una arquitectura de software que no solo es efectiva y sencilla de mantener, sino también altamente adaptable a posibles expansiones y mejoras en el futuro. Este enfoque conjunto ha contribuido de manera significativa a la solidez y flexibilidad de nuestro algoritmo de cálculos mecánicos, estableciendo una base robusta para el éxito continuo de nuestra aplicación.

2.4. Plan de pruebas

El propósito de llevar a cabo el plan de pruebas fue asegurar la calidad del prototipo desarrollado, así como validar cada uno de los requisitos funcionales y no funcionales definidos en el análisis propuesto por la metodología. A continuación, se describen las pruebas de integridad y las pruebas de aceptación y usabilidad realizadas en el actual proyecto.

2.4.1. Prueba de integridad

El propósito de estas pruebas consistió en evaluar si el prototipo de software había alcanzado un nivel de desarrollo avanzado. A través de este plan, se identificaron errores que afectaban la funcionalidad del software, los cuales fueron abordados de manera adecuada. En el enfoque de las pruebas de integridad, se llevaron a cabo evaluaciones de todo el software sin profundizar en detalles más específicos.

Estas pruebas se centraron en los casos de uso principales, los cuales son esenciales para el funcionamiento integral del software. Estos casos de uso son:

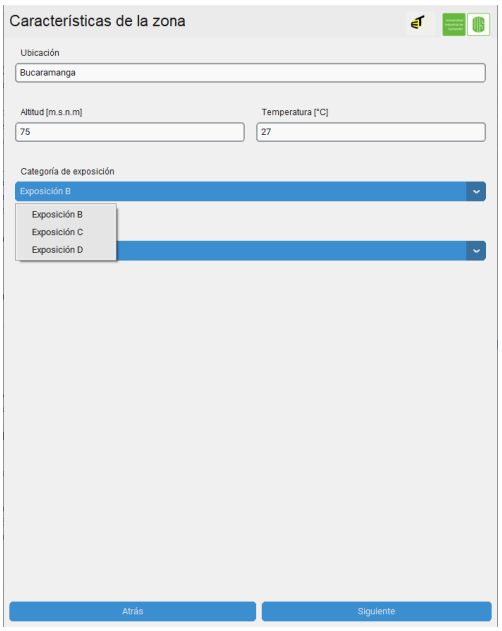
1. Permitir el ingreso de letras en los espacios indicados.

2. Permitir el ingreso de números en los espacios indicados.
3. El botón de atrás y siguiente funcionen de manera adecuada.
4. Indicativo de que falta un dato por llenar.
5. Listas desplegables funcionen de manera correcta.
6. Mostrar resultados finales.

Revisaremos las pruebas para identificar su correcto funcionamiento en las Figura 4, 5, 6.

Figura 5

Registro de caracteres tipo: cadena, enteros, flotantes y listas desplegables

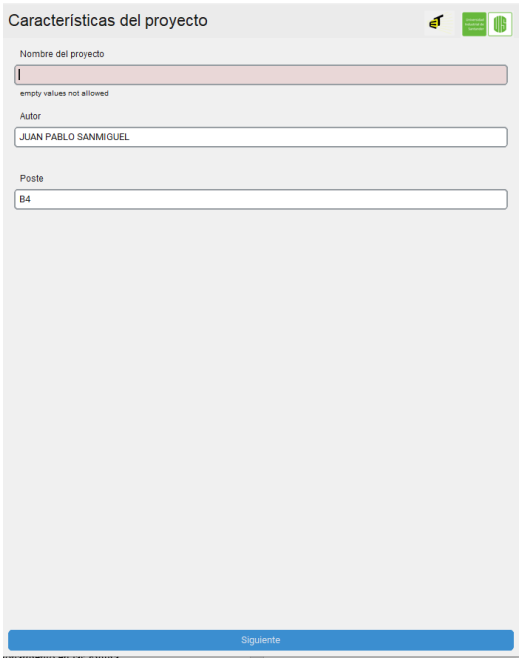


Nota. Tipo cadena hace referencia a todo dato de tipo texto, a su vez, tipo entero y flotante hace referencia a números.

Permite el ingreso de letras y números en las casillas correspondientes y las listas desplegables funciona mostrando todas las opciones.

Figura 6

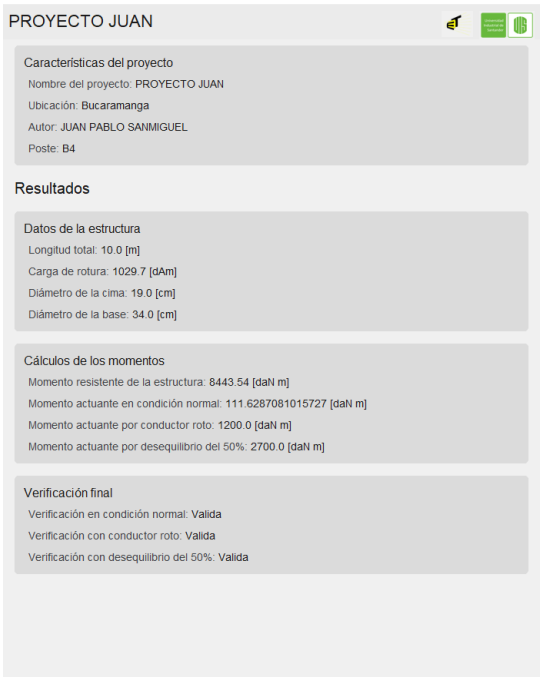
Comprobación de la acción de los botones e indicativo de casilla vacía



Permite devolver a la anterior pantalla dándonos a entender que funciona los botones de atrás y siguiente, del mismo modo, al tener una casilla sin datos nos la muestra en rojo indicando que falta llenarla.

Figura 7

Interfaz de resultados



En instancia final el software muestra los datos y resultados obtenidos del algoritmo. Con estas pruebas podemos asegurar que el software está ya en una etapa madura y funciona correctamente.

2.4.2. Prueba de aceptación funcional.

Para esta prueba realizaremos el ejemplo que se presenta en la guía base la cual es "Guía de cálculos mecánicos y selección de estructuras", en la figura 7 se presentan los datos del problema.

Figura 8

Datos ejemplo guía

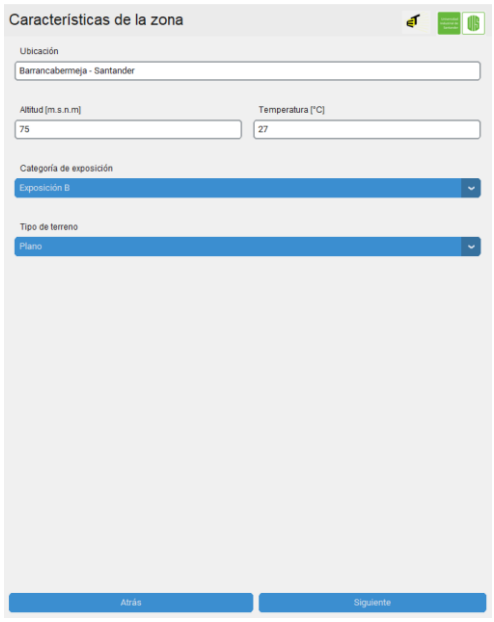
CARACTERÍSTICAS DE LA ZONA			
Ubicación	Barrancabermeja – Santander		
Altitud	75 m.s.n.m		
Temperatura media ambiente	27 °C		
Zona urbana sin efectos topográficos			
CARACTERÍSTICAS DE LA RED			
Nivel de tensión	Baja tensión		
Cambio de dirección de la línea	15°		
Vanos	Anterior	Posterior	
	25 m	25 m	
Conductores	2 x 4 (F) AWG + 4 (N) AWG		
Altura de los conductores	(N) 6.4 m	(F) 6.1 m	(F) 5.8 m
Diámetros conductores	6.360 mm		
Tensión mecánica en operación diaria	165.7113 daN		
Tensión mecánica en viento reducido	216.3053 daN		
Tensión mecánica en viento máximo	225.5590 daN		
Cantidad de aisladores	3		
Altura de los aisladores	(N) 6.4 m	(F) 6.1 m	(F) 5.8 m
Longitud de los aisladores	80 mm		
Diámetro de los aisladores	70 mm		

Nota. Tomado de *Guía de cálculos mecánicos y selección de estructuras* (p. 5), por Juan Arboleda, Jhoan Mejía, 2023

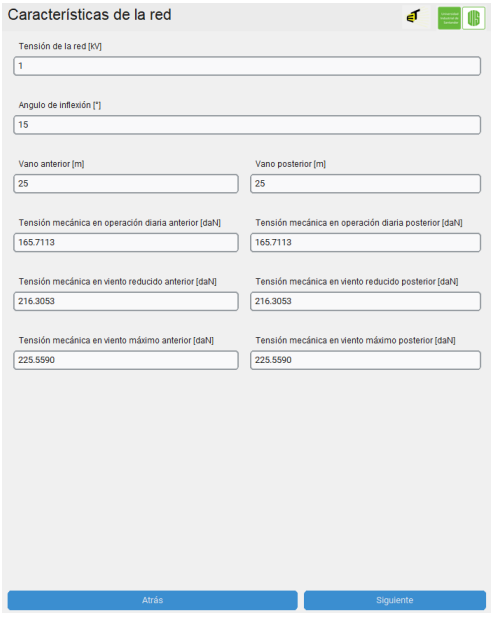
Se procede a ingresar los datos de las caracteristicas de la zona en el software, como se plasma en la figura 8.

Figura 9

Ingreso de datos al software



a) Paso 1: Ingreso datos características de la zona



b) Paso 2: Ingreso características de la red

c) Paso 3: Ingreso de las características de los conductores

d) Paso 4 Características de los aisladores

Una vez ingresado todos los datos requeridos por el software este procede a realizar los calculos internamente dando como producto final los resultados del tipo de poste a escoger, sus momentos y una verificación final, estos resultados se evidencian en la figura 9. Además se realiza una comparación con los datos suministrados en la "Guía para el cálculo y la selección de apoyos en estructuras de media y baja tensión para sistemas de distribución actualizado a normatividad vigente, RETIE y NTC2050" que se aprecian en la figura 10

Figura 10

Resultados obtenidos mediante el software

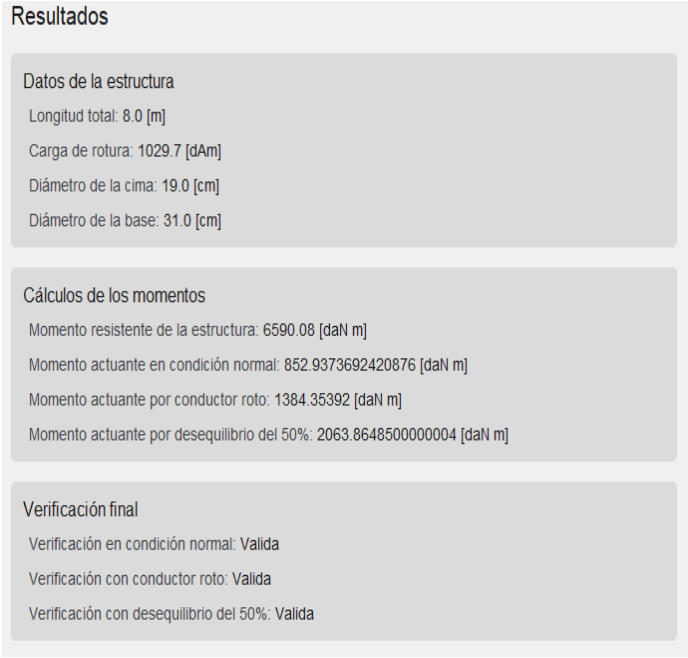
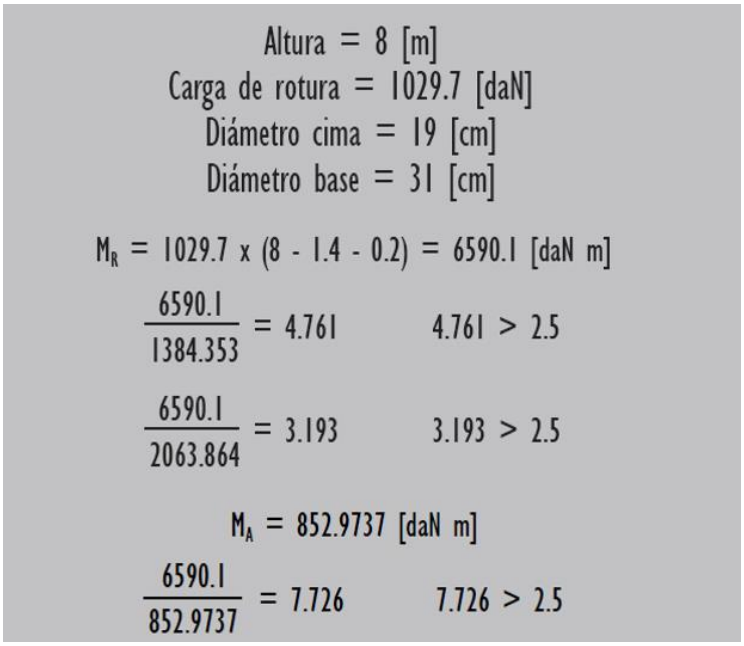


Figura 11

Resultados presentes en la guía



Nota. Tomado de *Guía de cálculos mecánicos y selección de estructuras* (p. 5), por Juan Arboleda, Jhoan Mejía, 2023

La prueba de aceptación funcional se llevó a cabo con éxito, y los resultados obtenidos de la guía de ejemplo confirman que el software se comporta de acuerdo con las expectativas y requisitos establecidos. La aplicación demostró de manera efectiva su capacidad para abordar el escenario planteado en la guía, proporcionando los resultados esperados y validando así su funcionalidad. Este proceso de prueba refuerza la confianza en la solidez y eficacia del software en cumplir con los objetivos planteados durante el desarrollo.

3. Conclusiones

La culminación de este proyecto de tesis marca un hito significativo en nuestro recorrido académico y profesional al lograr el desarrollo exitoso de un software en Python basado en la guía de cálculos mecánicos y selección de estructuras de concreto. A lo largo de esta investigación, se ha abordado de manera integral el desafío de ofrecer una solución práctica y eficiente destinada a ingenieros eléctricos, estudiantes de ingeniería eléctrica o a cualquier usuario con conocimiento sobre tema que desee usarlo.

La implementación de la metodología Scrum, con su enfoque ágil y colaborativo, ha permitido la adaptabilidad y la entrega incremental, asegurando un proceso de desarrollo sólido y orientado a resultados tangibles. La elección de la biblioteca tkinter para la interfaz de usuario y la aplicación de la metodología Atomic Design en la estructuración del algoritmo no solo resultaron ser decisiones acertadas, sino que también contribuyeron a la usabilidad, modularidad y eficacia general del software.

Las pruebas realizadas, incluyendo pruebas de integridad y pruebas de aceptación funcional, no solo confirmaron la calidad del software, sino que validaron su capacidad para abordar escenarios específicos según la guía de cálculos establecida. El software no solo cumple con los requisitos funcionales y no funcionales, sino que también proporciona una interfaz de fácil entendimiento y que cualquier usuario pueda usar.

Este proyecto no solo representa el logro de los objetivos académicos propuestos, sino que también destaca la aplicación exitosa de principios de ingeniería, programación y metodologías ágiles. Se espera que esta herramienta facilite el trabajo específico de los profesionales en ingeniería eléctrica y sirva como ejemplo del potencial de la integración efectiva de la tecnología y la ingeniería para abordar desafíos prácticos en este campo específico.

Es importante destacar que, en futuras versiones, se contempla la posibilidad de ampliar la funcionalidad del software incorporando cálculos para otros tipos de postes, más allá de los de concreto. Este enfoque evolutivo busca abarcar un espectro más amplio de necesidades, mejorando así la versatilidad y utilidad de la herramienta en diversos contextos de ingeniería. Este proyecto no solo representa un logro actual, sino también sienta las bases para futuras expansiones que seguirán optimizando la utilidad del software.

Referencias Bibliográficas

Alabi, V. (2017). Matlab GUI as Assistance Tool for Transmission. *IEEE Central America and Panama Student Conference*, (págs. 1-6). Panama. doi:10.1109/CONESCAPAN.2017.8277594.

Arboleda, J., & Jhoan, M. (2023). *Guía para el cálculo y la selección de apoyos en estructuras de media y baja tensión para sistemas de distribución actualizado a normatividad vigente, RETIE y NTC2050 [Tesis de pregrado, Universidad Industrial de Santander].* Repositorio Institucional. Obtenido de https://noesis.uis.edu.co/handle/20.500.14071/12282

Frost, B. (2016) Atomic Design. Brad Frost. Obtenido de https://samatharavinda.lk/img/books/Atomic%20Design%20(%20samatharavindalk%20).pdf

Fernández Montoro, A. (2013). Orientación a objetos. En *Python 3 al descubierto* (2da ed., págs. 100-134). Alfaomega Grupo Editor.

Guagliano, C. (2019). *Programación en Python- Vol II.* Obtenido de https://books.google.es/books?hl=es&lr&id=y1yzDwAAQBAJ&oi=fnd&pg=PA5&dq=programaci%C3%B3n+orientada+a+objetos+en+Python+&ots=G0FTDB-F7c&sig=4sFC6KGAORPfkpKbNc3-L16XQuw&pli=1#v=onepage&q=programaci%C3%B3n%20orientada%20a%20objetos%20en%20Python&f=false

Pardo, C., Suescún, E., Jojoa, H., Zambrano, R., & Ortega, W. (2020). Modelo de referencia para la adopción e implementación de Scrum en la industria de software.

Investigación e Innovación en Ingenierías, 8(3), 14-28. doi:https://doi.org/10.17081/invinno.8.3.4700

Rodríguez, C., & Dorado, R. (2015). ¿Por qué implementar Scrum? *ONTARE, 3*(1), 125-144. Obtenido de https://dialnet.unirioja.es/servlet/articulo?codigo=8705520

Anexos

Anexo a.

Nota**.** Al momento de realizar la descarga del ejecutable, puede suceder que se muestre un anuncio con lo siguiente “archivo sospechoso” o a su vez ser marcado como un virus, esto se debe a que el aplicativo carece de una firma certificada. Por favor hacer caso omiso.

Manual de uso

Antes de utilizar el software, se requiere recopilar información esencial para llevar a cabo los cálculos y la selección de estructuras de manera precisa. Esto incluye datos relacionados con las características de la zona, tales como altitud, temperatura y tipo de terreno, así como la ubicación geográfica específica de la estructura en cuestión. Además, se necesita información detallada acerca de la red, como el nivel de tensión en kilovoltios (kV), las distancias del vano anterior y posterior, un análisis previo de los conductores para evaluar sus tensiones mecánicas en diversas situaciones, y el ángulo de inflexión.

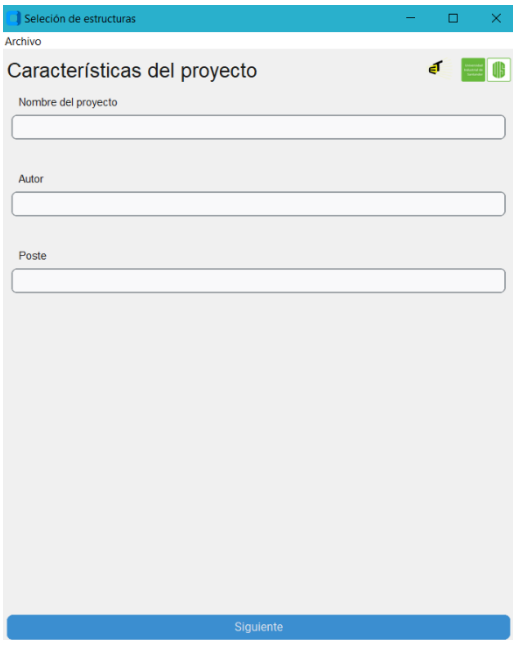
Posteriormente, se requiere información específica sobre los conductores y aisladores que serán utilizados en el proyecto. Esto implica conocer el número de conductores y aisladores a emplear, así como sus respectivas alturas, diámetros y longitudes. Una vez tengamos todos la información anterior mencionada procedemos iniciar nuestro software.

Paso 1.

Al iniciar el software, la primera ventana solicitará información esencial del usuario para la identificación del proyecto. Esto incluye detalles como el nombre del proyecto, el nombre del autor y la denominación específica del poste bajo estudio.

Figura 12

Ventana 1. Características del proyecto

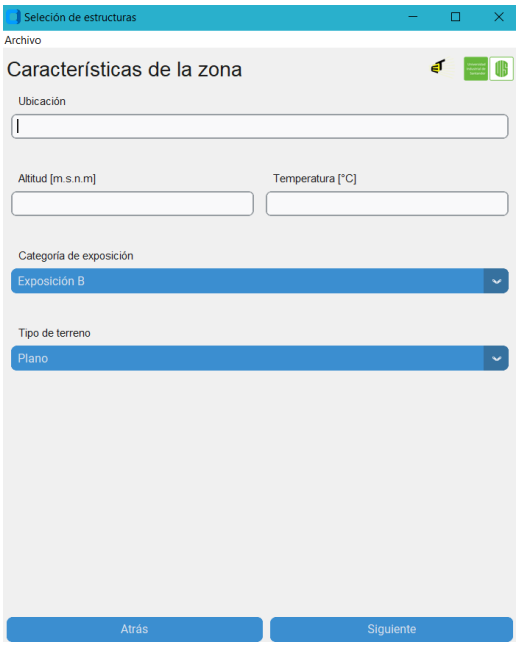


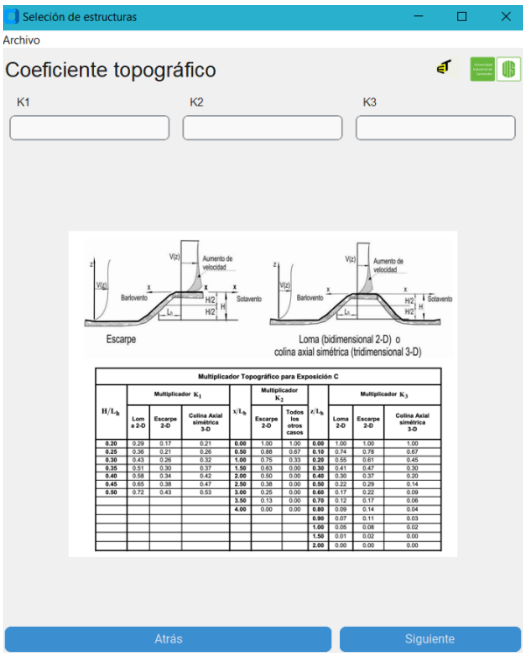
Paso 2.

Tras pulsar "Siguiente", se nos pedirá ingresar datos sobre la ubicación geográfica, tales como el nombre de la zona, su altitud y temperatura. Además, se especificará la exposición del poste, clasificada como B, C o D, en el caso de ser exposición C y seleccionar montañoso, se solicitarán ciertas constantes basadas en una tabla incorporada que considera el tipo de montaña y las magnitudes físicas asociadas a través de gráficas.

Figura 13

Ventana 2. Características de la zona



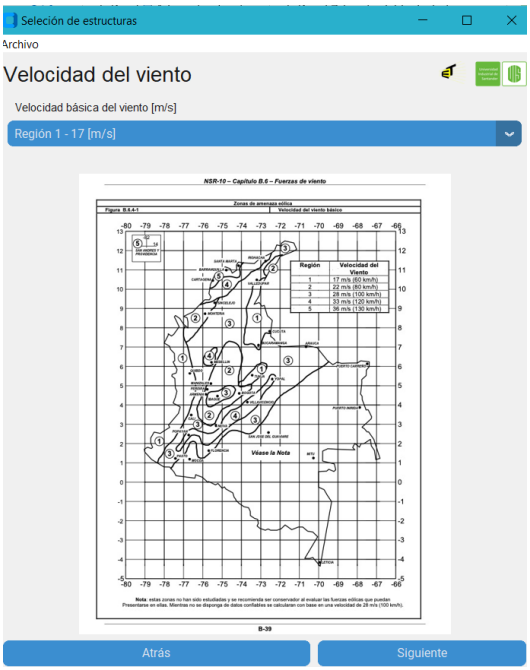


Paso 3.

Luego al dar "Siguiente", se visualizará un mapa de Colombia donde se deberá seleccionar la región correspondiente al emplazamiento del poste para determinar la velocidad básica del viento en esa área.

Figura 14

Ventana 3. Velocidad del viento

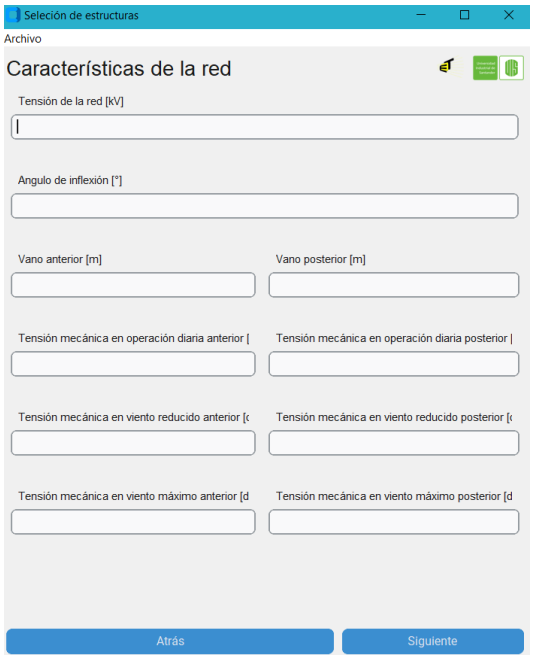


Paso 4.

La siguiente ventana se enfoca en las características de la red eléctrica, solicitando datos como la tensión eléctrica, el ángulo de inflexión y la distancia entre vanos, junto con sus correspondientes tensiones mecánicas, indicando las unidades adecuadas para la entrada de datos.

Figura 15

Ventana 4. Características de la red

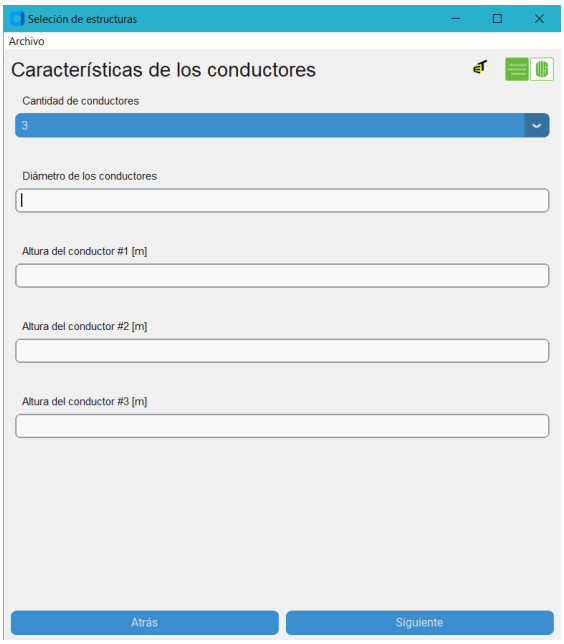


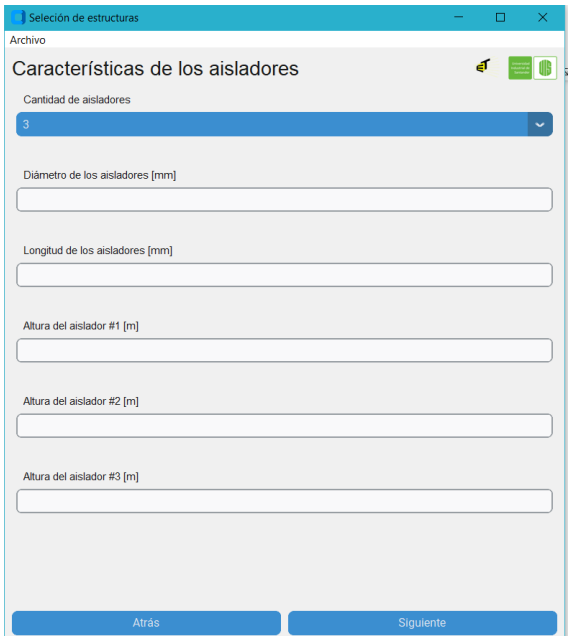
Paso 5

Al avanzar, se accede a la ventana de características de conductores, donde se ingresan datos sobre la cantidad, diámetro y altura de cada conductor, el proceso continúa con la introducción de datos similares para los aisladores en la ventana subsiguiente.

Figura 16

Ventana 5-6. Características de conductores y aisladores



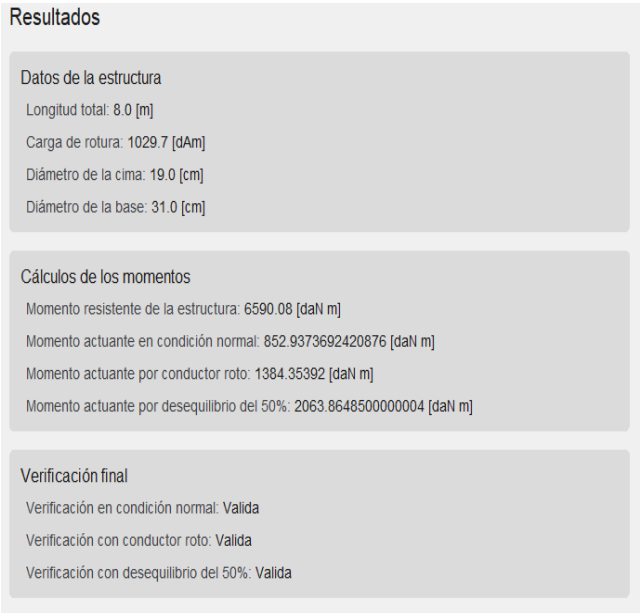


Paso 6

Al pulsar "Siguiente", el software realiza los cálculos necesarios y presenta la ventana final de resultados. Además, se incorpora una opción de archivo en la parte superior izquierda, donde es posible cerrar el software o visualizar los autores de este.

Figura 17

Ventana 7. Resultados obtenidos



Anexo b.

Link y QR Software.

A continuación, se presentará el link y el QR de “OneDrive” en el cual encuentra la guía.

Link:

https://correouisedu-my.sharepoint.com/:f:/g/personal/brayan_quiroga1_correo_uis_edu_co/Et3M-_rVXFVOrvEidWKcuIQBowtTLtkncfLw6z8tC_WOCg?e=oJbBJm

Link video tutorial uso del software:

https://drive.google.com/file/d/1-AxprwAxUgBOWZEleaMKMOXvtEvk--zc/view?usp=drive_link

QR: