# Documentación Ejecutiva del Proyecto CirKit

**Simulador de Circuitos Eléctricos por Diseño**

**Versión:** 1.0.0

**Fecha:** Noviembre 2025

Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

**Curso:** Administración de Proyectos

**Equipo:** C09676 Studios

---

# Resumen Ejecutivo

CirKit es un simulador de circuitos eléctricos desarrollado en Python 3.13.7 como proyecto académico para el curso de Administración de Proyectos. La aplicación permite realizar cálculos fundamentales en circuitos eléctricos básicos, incluyendo la aplicación de la Ley de Ohm, análisis de resistencias en serie y paralelo, y cálculo de potencia eléctrica.

El proyecto implementa una arquitectura modular que separa la lógica de negocio de la interfaz de usuario, permitiendo múltiples interfaces (consola, Tkinter, Kivy) que comparten el mismo núcleo de cálculo. Esta documentación técnica describe la arquitectura del sistema, los componentes principales, la metodología de desarrollo utilizada, las decisiones de diseño tomadas y las guías para el uso, mantenimiento y extensión del software.

**Palabras clave:** Simulación de circuitos, Python, Ley de Ohm, Tkinter, Kivy, análisis de resistencias, potencia eléctrica, educación en ingeniería.

---

# 1. Introducción

## 1.1 Contexto del Proyecto

El estudio de circuitos eléctricos constituye una base fundamental en la formación de ingenieros y técnicos en áreas relacionadas con la electricidad y la electrónica. La comprensión de conceptos como la Ley de Ohm, el comportamiento de resistencias en diferentes configuraciones y el cálculo de potencia eléctrica es esencial para el análisis y diseño de sistemas eléctricos y electrónicos.

Tradicionalmente, los estudiantes aprenden estos conceptos mediante ejercicios manuales que pueden resultar repetitivos y propensos a errores de cálculo. Las herramientas de simulación comerciales, si bien potentes, suelen ser complejas para usuarios principiantes y requieren una curva de aprendizaje considerable.

CirKit surge como respuesta a la necesidad de contar con una herramienta educativa accesible, de código abierto y específicamente diseñada para el aprendizaje de los fundamentos de análisis de circuitos eléctricos. El proyecto fue desarrollado como parte del curso de Administración de Proyectos en la Benemérita Universidad Autónoma de Puebla durante el semestre de otoño 2025.

## 1.2 Objetivos del Sistema

### 1.2.1 Objetivo General

Desarrollar un simulador de circuitos eléctricos educativo que permita a estudiantes y profesores realizar cálculos fundamentales de circuitos de forma rápida, precisa y accesible, promoviendo el aprendizaje de los conceptos básicos de electricidad.

### 1.2.2 Objetivos Específicos

1. Implementar algoritmos precisos para el cálculo de parámetros eléctricos basados en la Ley de Ohm, resistencias en serie y paralelo, y potencia eléctrica.
2. Diseñar una arquitectura modular que permita la extensión y mantenimiento sencillo del software.
3. Desarrollar múltiples interfaces de usuario (consola, Tkinter, Kivy) para adaptarse a diferentes preferencias y contextos de uso.
4. Crear documentación técnica y de usuario completa que facilite el uso, mantenimiento y extensión del sistema.

5. Aplicar metodologías de administración de proyectos y control de versiones en un entorno colaborativo.
6. Validar la precisión de los cálculos mediante casos de prueba y comparación con valores teóricos conocidos.

#### 1.2.2.1 KPI de Objetivo General

**Nombre del KPI:** Funcionalidad Básica del Simulador

**Función del KPI:** Validar

**Objetivo y Cronograma**: Fiabilidad de los cálculos para circuitos ideales y reales

**Unidad de Medida:** % Fiabilidad

**Solicitado por:** Razo Montañez Gael Askary

**Ejecutado por:** Garcia Vera Oscar Uriel

**Descripción:**

Este KPI se encarga de comprobar que el simulador sea capaz de realizar cálculos correctos en diferentes tipos de circuitos, asegurando que los resultados obtenidos sean precisos y congruentes. Su propósito es garantizar que la herramienta funcione adecuadamente desde su base operativa. Además, ayuda a detectar inconsistencias en los algoritmos y mejora la confianza del usuario final. Este control es esencial para validar la calidad técnica del simulador antes de su uso avanzado.

#### 1.2.2.2 KPI Objetivo Específico 1

**Nombre:** Tiempo de Implementación de Componentes UI

**Función:** Optimizar

**Objetivo:** Reducir tiempo de desarrollo UI en 30%

**Unidad de Medida:** Horas hombre

**Solicitado por:** Lopez Momox Limhi Gerson

**Ejecutado por:** Huerta Maldonado Arturo

**Descripción:**

Este KPI mide el tiempo que se tarda en construir e implementar los elementos visuales de la interfaz del simulador. Su enfoque es mejorar la eficiencia del desarrollo, reduciendo tiempos y eliminando procesos innecesarios. Permite identificar cuellos de botella en la creación de la UI y aplicar estrategias que agilicen el flujo de trabajo. Gracias a esto, se busca lograr un desarrollo más rápido sin afectar la calidad visual ni funcional.

#### 1.2.2.3 KPI Objetivo Específico 2

**Nombre:** Índice de Estabilidad de Componentes

**Función:** Verificar

**Objetivo:** 5 fallas en 100 simulaciones consecutivas (Semanas 2–3 Nov)

**Unidad de Medida:** Tasa de estabilidad

**Solicitado por:** Garcia Vera Oscar Uriel

**Ejecutado por:** Lopez Momox Limhi Gerson

**Descripción:**

Este KPI se enfoca en evaluar qué tan estables son los componentes del simulador durante su ejecución continua. Mide la cantidad de fallas que se presentan en una serie de pruebas repetidas, lo cual permite entender la confiabilidad real del sistema. Su propósito es identificar debilidades que puedan afectar el uso prolongado del programa. Con ello, se asegura que el simulador mantenga un rendimiento consistente incluso en escenarios exigentes.

#### 1.2.2.4 KPI Objetivo Específico 3

**Nombre:** Velocidad de Resolución de Incidentes Críticos

**Función:** Mejorar

**Objetivo:** 90% de errores críticos resueltos en < 24 horas (Semanas 2–3 Nov)

**Unidad de Medida:** Horas y Porcentaje

**Solicitado por:** Gutierrez Hernandez Juan Enrique

**Ejecutado por:** Razo Montañez Gael Askary

**Descripción:**

Este KPI mide la rapidez con la que el equipo responde y soluciona fallas graves que puedan comprometer el funcionamiento del simulador. Su objetivo es mantener un tiempo de reacción corto para evitar interrupciones o riesgos en el desarrollo. Ayuda a evaluar qué tan efectiva es la comunicación interna y la capacidad de resolución técnica del equipo. Además, impulsa la mejora continua en la gestión de incidentes críticos.

#### 1.2.2.5 KPI de Desempeño

**Nombre:** Desviación Acumulada en Hitos Clave

**Función:** Monitorear

**Objetivo:** Máximo 5 días de desviación total en hitos críticos (Agosto–Noviembre)

**Unidad de Medida:** Días de desviación

**Solicitado por:** Huerta Maldonado Arturo

**Ejecutado por:** Gutierrez Hernandez Juan Enrique

**Descripción:**

Este KPI monitorea qué tanto se desvía el proyecto de las fechas previstas para los hitos más importantes. Su función es asegurar que el calendario de trabajo se cumpla de manera razonable y sin retrasos significativos. Permite detectar problemas de planificación, carga de trabajo o ejecución temprana. También ayuda a ajustar tiempos y recursos para mantener el avance del proyecto dentro del margen esperado.

# 1.3 Alcance y Limitaciones

Esta sección define los límites funcionales y las capacidades específicas que incluye la versión actual del simulador CirKit. Establece claramente qué funcionalidades están disponibles para el usuario y cuáles quedan excluidas del sistema en esta etapa de desarrollo. La delimitación precisa del alcance permite establecer expectativas realistas sobre las capacidades del software y proporciona una base sólida para futuras expansiones y mejoras del proyecto educativo.

## 1.3.1 Alcance

El sistema CirKit versión 1.0 incorpora funcionalidades fundamentales para el análisis básico de circuitos eléctricos, dirigido principalmente a estudiantes y entusiastas de la electrónica. La aplicación permite realizar cálculos esenciales como la Ley de Ohm y combinaciones de resistencias, ofreciendo tres interfaces diferentes para adaptarse a las preferencias del usuario. La arquitectura modular garantiza un mantenimiento sencillo y futuras expansiones del sistema, mientras que la validación de entrada asegura la confiabilidad de los resultados calculados.

### 1.3.2 Limitaciones

La versión actual del sistema presenta limitaciones inherentes a su diseño como herramienta educativa básica, excluyendo componentes avanzados como capacitores e inductores de su capacidad de análisis. El sistema se concentra exclusivamente en circuitos de corriente continua con configuraciones resistivas simples, sin incluir análisis de frecuencia o comportamiento transitorio. Tampoco incorpora funcionalidades de visualización gráfica de circuitos ni métodos de análisis complejos como mallas o nodos para múltiples componentes. Estas limitaciones representan oportunidades de crecimiento para versiones futuras del software educativo.

## 1.4 Audiencia Objetivo

Esta documentación está dirigida a:

1. **Estudiantes de ingeniería que podrían utilizar** la aplicación como herramienta de aprendizaje.
2. **Profesores e instructores** que deseen integrar la herramienta en sus cursos de circuitos eléctricos.
3. **Desarrolladores** interesados en contribuir al proyecto o extender sus funcionalidades.
4. **Administradores de sistemas** responsables del despliegue de la aplicación en laboratorios educativos.
5. **Investigadores** en educación en ingeniería que estudien herramientas de aprendizaje asistido por computadora.

# 2. Fundamentos Teóricos

Esta sección presenta los fundamentos teóricos de los cálculos implementados en CirKit, proporcionando el contexto matemático y físico necesario para comprender el funcionamiento del sistema.

## 2.1 Ley de Ohm

La Ley de Ohm, formulada por Georg Simon Ohm en 1827, establece la relación fundamental entre voltaje, corriente y resistencia en un conductor eléctrico. Esta ley es expresada matemáticamente como:

$V = I \cdot R$

Donde:

- $V$ es el voltaje o diferencia de potencial (en voltios, V).
- $I$ es la corriente eléctrica (en amperios, A).
- $R$ es la resistencia eléctrica (en ohmios, Ω).

De esta ecuación fundamental se derivan las siguientes relaciones:

$I = \frac{V}{R}\ y\ R = \frac{V}{I}$

### 2.1.1 Aplicación en CirKit

El módulo de Ley de Ohm en CirKit permite al usuario calcular cualquiera de las tres variables cuando se conocen las otras dos. El sistema implementa validaciones para evitar divisiones por cero y valores no físicos (negativos).

**Ejemplo de cálculo:**

- Voltaje conocido: $V = 12\ V$
- Resistencia conocida: $R = \frac{V}{I} = 4\ \Omega$
- Corriente calculada: $I = \frac{V}{R} = \frac{12}{4} = 3\ A$

## 2.2 Resistencias en Serie

Cuando dos o más resistencias se conectan en serie, la corriente que circula por cada una es la misma, y el voltaje total aplicado se divide entre las resistencias. La resistencia total o equivalente de resistencias en serie se calcula como:

$$R_{total} = R_1 + R_2 + R_3 + \cdots + R_n = \sum_{i=1}^{n} R_i$$

### 2.2.1 Propiedades de las Resistencias en Serie

1. La corriente es la misma en todos los componentes: $I_{total} = I_1 = I_2 = \cdots = I_n$.
2. El voltaje total es la suma de los voltajes parciales: $V_{total} = V_1 + V_2 + \cdots + V_n$.
3. La resistencia total es siempre mayor que la resistencia individual más grande.
4. La adición de más resistencias en serie aumenta la resistencia total.

**Ejemplo de cálculo:**

- Resistencias: $R_1 = 10\ \Omega$, $R_2 = 20\ \Omega$, $R_3 = 30\ \Omega$
- Resistencia total: $R_{total} = \sum_{i=1}^{n} R_i = 10 + 20 + 30 = 60\ \Omega$

## 2.3 Resistencias en Paralelo

Cuando las resistencias se conectan en paralelo, el voltaje en cada una es el mismo, mientras que la corriente total se divide entre las resistencias. La resistencia equivalente de resistencias en paralelo se calcula mediante:

$$\frac{1}{R_{total}} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \cdots + \frac{1}{R_n} = \sum_{i=1}^{n} \frac{1}{R_i}$$

De donde se obtiene:

$$R_{total} = \frac{1}{\sum_{i=1}^{n} \frac{1}{R_i}}$$

Para el caso particular de dos resistencias en paralelo, la fórmula se simplifica a:

$$R_{total} = \frac{R_1 \cdot R_2}{R_1 + R_2}$$

## 2.3.1 Propiedades de las Resistencias en Paralelo

1. El voltaje es el mismo en todos los componentes: $V_{total} = V_1 = V_2 = \cdots = V_n$.
2. La corriente total es la suma de las corrientes parciales: $I_{total} = I_1 + I_2 + \cdots + I_n$.
3. La resistencia total es siempre menor que la resistencia individual más pequeña.
4. La adición de más resistencias en paralelo disminuye la resistencia total.

**Ejemplo de cálculo:**

- Resistencias: $R_1 = 10\ \Omega$, $R_2 = 20\ \Omega$
- Resistencia total: $R_{total} = \frac{R_1 \cdot R_2}{R_1 + R_2} = \frac{10 \cdot 20}{10 + 20} = \frac{200}{30} \approx 6.67\ \Omega$

# 3. Arquitectura del Sistema

Es la estructura general que organiza cómo funciona un sistema o software. Define sus partes principales y cómo se comunican entre sí. Ayuda a que el sistema sea más fácil de mantener, escalar y entender. Es como el mapa de alto nivel del proyecto.

## 3.1 Diseño Arquitectónico

Proceso de planear y decidir cómo se construirá el sistema antes de programarlo. Su objetivo es asegurar que el software sea eficiente, seguro y fácil de mejorar. CirKit implementa una arquitectura de software modular basada en el patrón de diseño de separación de responsabilidades (Separation of Concerns). La arquitectura se divide en tres capas principales:

1. **Capa de Lógica de Negocio:** Contiene los algoritmos de cálculo y la lógica del dominio.
2. **Capa de Presentación:** Implementa las interfaces de usuario.
3. **Capa de Integración:** Coordina la comunicación entre la lógica y las interfaces.

Esta separación permite que el núcleo de cálculo sea completamente independiente de las interfaces de usuario, facilitando la adición de nuevas interfaces sin modificar la lógica de negocio.

## 3.2 Modelo de Capas

Es una forma de organizar el software en capas, donde cada uno tiene una función específica. Las capas superiores usan los servicios de las inferiores, pero no al revés. Esto permite ordenar el código y evitar dependencias desordenadas. Facilita el mantenimiento y los cambios sin afectar a todo el sistema.

| Capa | Componentes | Responsabilidad |
|---|---|---|
| Presentación | ejecutable.py<br>interfaz_tkinter.py<br>interfaz_kivy.py | Interfaz de consola<br>Interfaz gráfica Tkinter<br>Interfaz gráfica Kivy |
| Lógica de Negocio | logica_circuitos.py | Algoritmos de cálculo<br>Validación de datos<br>Funciones matemáticas |
| Datos | Variables locales<br>Estructuras de datos | Almacenamiento temporal<br>Organización de resultados |

*Tabla 1: Arquitectura de capas del sistema CirKit*

## 3.3 Diagrama de Componentes

El diagrama representa módulos, bases de datos, servicios o aplicaciones que colaboran entre sí. Sirve para visualizar la estructura técnica sin entrar en detalles de código. sistema está compuesto por los siguientes módulos principales:

**Módulo de Lógica (logica_circuitos.py):**

- Funciones de cálculo de la Ley de Ohm.
- Funciones de resistencias en serie.
- Funciones de resistencias en paralelo.
- Utilidades de validación.

**Módulo de Consola (ejecutable.py):**

- Menú principal de navegación.
- Entrada de datos por teclado.
- Visualización de resultados en texto.
- Manejo de errores de entrada.

**Módulo Tkinter (interfaz_tkinter.py):**

- Ventanas de aplicación GUI.
- Widgets de entrada (Entry, Button).
- Widgets de salida (Label).
- Gestión de eventos.

**Módulo Kivy (interfaz_kivy.py):**

- Layouts multiplataforma.
- Widgets touch-friendly.
- Archivos .kv para diseño declarativo.
- Portabilidad móvil.

## 3.4 Patrones de Diseño Aplicados

Son soluciones comunes y probadas para problemas frecuentes en el desarrollo de software. Ayudan a ordenar el código, hacerlo más reutilizable y evitar errores. Se aplican para mejorar la calidad y organización del sistema.

### 3.4.1 Patrón MVC (Modelo-Vista-Controlador)

CirKit adopta una variante simplificada del patrón MVC:

- **Modelo:** logica_circuitos.py - Contiene la lógica de negocio y cálculos.
- **Vista:** interfaz_*.py - Presenta la información al usuario.
- **Controlador:** Funciones de integración en cada interfaz que coordinan modelo y vista.

### 3.4.2 Patrón de Módulos

Cada archivo Python representa un módulo independiente con responsabilidades bien definidas, permitiendo:

- Reutilización de código.
- Facilidad de prueba unitaria.
- Mantenimiento simplificado.
- Extensibilidad del sistema.

### 3.4.3 Principio DRY (Don't Repeat Yourself)

La lógica de cálculo se implementa una sola vez en logica_circuitos.py y es reutilizada por todas las interfaces, evitando duplicación de código.

# 4. Especificación Técnica

La sección proporciona información detallada sobre todos los componentes necesarios para instalar y ejecutar correctamente el simulador de circuitos. Se incluyen requisitos de hardware, software y las tecnologías utilizadas en el desarrollo del proyecto para garantizar un funcionamiento óptimo del sistema.

## 4.1 Requisitos del Sistema

Esta parte enumera los componentes físicos y programas esenciales que el equipo de cómputo debe tener para que la aplicación funcione sin problemas. Los requisitos se dividen en hardware para componentes tangibles y software para programas necesarios, facilitando la verificación por parte del usuario.

### 4.1.1 Requisitos de Hardware

| **Componente** | **Especificación Mínima** |
|---|---|
| Procesador | Intel Core i3 o equivalente |
| Memoria RAM | 2 GB |
| Espacio en disco | 100 MB |
| Pantalla | 1024x768 píxeles |
| Dispositivo de entrada | Teclado y ratón |

*Tabla 2: Requisitos mínimos de hardware*

Los requisitos de hardware representan los componentes físicos mínimos que el equipo necesita para ejecutar el programa eficientemente. Estos elementos garantizan que no haya lentitud durante las simulaciones e incluyen capacidad de procesamiento, memoria suficiente y espacio de almacenamiento adecuado para la instalación completa del software.

### 4.1.2 Requisitos de Software

| **Software** | **Versión** |
|---|---|
| Sistema Operativo | Windows 10/11 |
| Python | 3.13.7 o superior |
| pip | Última versión estable |
| Git | 2.30 o superior (opcional) |

*Tabla 3: Requisitos de software*

Los requisitos de software constituyen los programas base que deben estar preinstalados en el sistema operativo antes de utilizar el simulador. Esta lista asegura la compatibilidad con las tecnologías utilizadas en el desarrollo y facilita el proceso de instalación para el usuario final del sistema.

# 4.2 Tecnologías Utilizadas

Este apartado describe el conjunto de herramientas y lenguajes de programación seleccionados para desarrollar una aplicación educativa eficiente. Cada tecnología fue elegida por sus ventajas particulares en rendimiento, facilidad de uso y compatibilidad con los objetivos del proyecto educativo.

## 4.2.1 Lenguaje de Programación

## Python 3.13.7

Python fue seleccionado por las siguientes razones:

- Sintaxis clara y legible, ideal para proyectos educativos.
- Amplia disponibilidad de bibliotecas para GUI.
- Multiplataforma sin modificaciones de código.
- Facilidad de mantenimiento y extensión.
- Comunidad activa y documentación extensa.

El lenguaje de programación Python fue seleccionado por su sintaxis clara y fácil comprensión, ideal para proyectos académicos. Permite escribir código legible y mantenible, además de contar con una amplia comunidad de apoyo y numerosas bibliotecas especializadas para diferentes funcionalidades requeridas.

## 4.2.2 Bibliotecas y Frameworks

Las bibliotecas y frameworks representan conjuntos de herramientas pre desarrolladas que aceleran la creación de interfaces gráficas. Estas herramientas permiten a los desarrolladores concentrarse en la lógica de la aplicación sin crear cada componente desde cero, mejorando la productividad del desarrollo.

**Tkinter**

- Biblioteca estándar de Python para GUI.
- No requiere instalación adicional.
- Ideal para interfaces de escritorio sencillas.
- Widgets nativos del sistema operativo.

La biblioteca Tkinter está incluida por defecto con Python para crear interfaces gráficas simples utilizando componentes nativos. Proporciona una curva de aprendizaje suave y es ideal para aplicaciones de escritorio tradicionales con requisitos de interfaz básicos pero completamente funcionales.

**Kivy**

- Framework multiplataforma para GUI modernas.
- Soporte para aplicaciones móviles (Android/iOS).
- Interfaz touch-friendly.
- Lenguaje KV para diseño declarativo.

El framework Kivy está especializado en interfaces visuales avanzadas que funcionan en múltiples plataformas incluyendo dispositivos móviles. Ofrece capacidades táctiles nativas y un lenguaje de diseño propio que facilita la creación de interfaces atractivas y responsivas para los usuarios.

**Bibliotecas estándar de Python:**

- os - Maneja operaciones con archivos y carpetas del sistema operativo
- sys - Controla configuraciones y parámetros internos del programa en ejecución
- typing - Define tipos de datos para mejor documentación y validación del código fuente

## 4.3 Estructura del Proyecto

La estructura del proyecto muestra la organización jerárquica de archivos y carpetas que mantiene el código fuente ordenado. Esta organización separa claramente los diferentes componentes como documentación, código fuente, pruebas y recursos visuales para mejor mantenimiento del sistema.

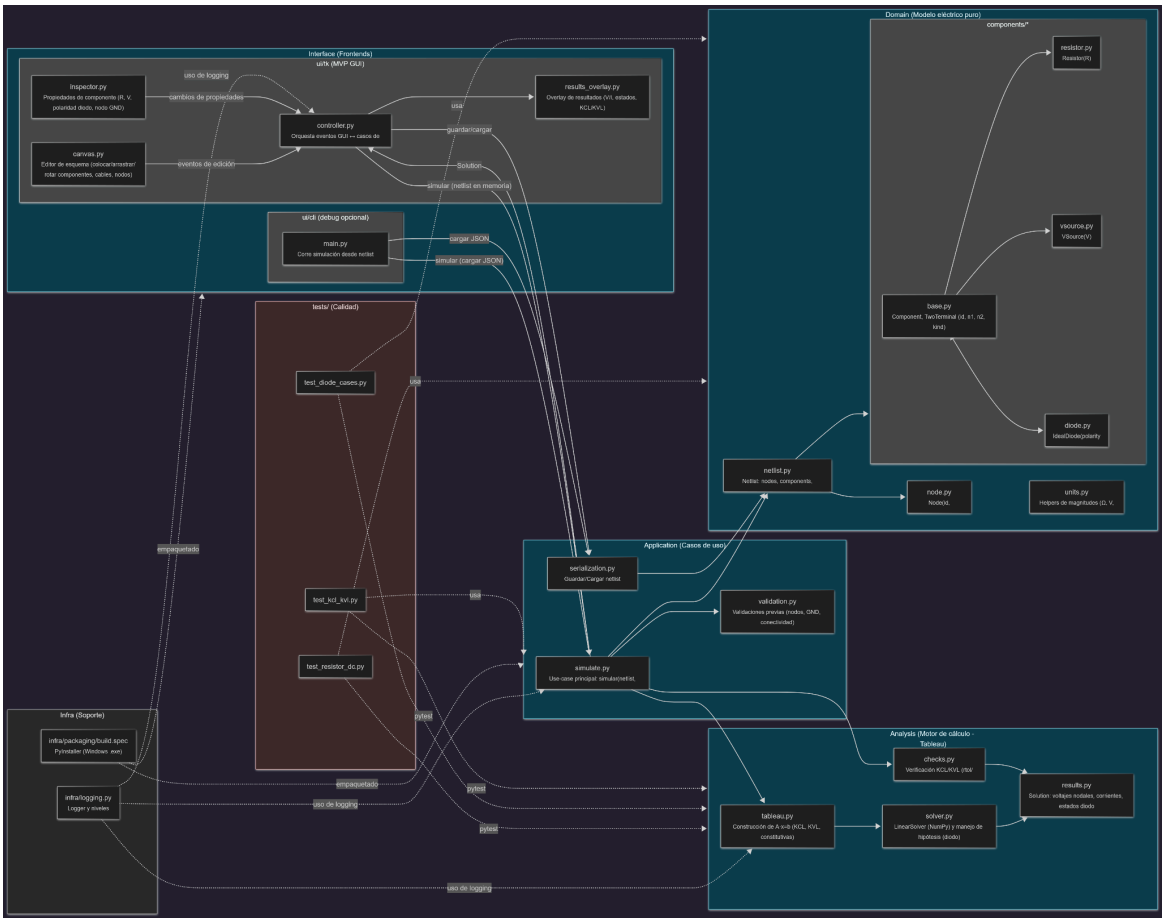



*Imagen 1. Diagrama de paquetes sobre la estructura general del proyecto.*

## 4.3.1 Descripción de la Arquitectura de Software

### 1. Interfaz (API y CLI)

Esta capa funciona como punto de entrada del sistema. Aquí se reciben los parámetros, configuraciones y comandos del usuario. Su responsabilidad es validar la entrada y convertirla en instrucciones comprensibles para las capas internas.

Componentes principales:

- **run.py**: Administra la ejecución del sistema y controla parámetros como el modo de operación, los archivos de entrada o las opciones seleccionadas.
- **main.py**: Procesa argumentos, controla el flujo principal y orquesta las llamadas hacia los módulos encargados de los cálculos.

- **Esquemas JSON**: Validan archivos de entrada y definen estructuras estándar para la configuración.
- **conversion.py y campos.py**: Ejecutan tareas de parsing, transformación y validación de propiedades y unidades.

**2. Dominio (Modelo Científico Principal)**

Esta capa contiene la lógica central del negocio, especialmente los modelos matemáticos y científicos utilizados para ejecutar los cálculos fundamentales.

Componentes principales:

- **material.py**: Gestiona propiedades y registros de materiales.
- **visor.py**: Implementa el modelo matemático o físico que sustenta los cálculos principales.
- **cálculo.py**: Realiza operaciones numéricas críticas, tales como interpolaciones y ecuaciones avanzadas.
- **tupper.py**: Ejecuta cálculos específicos asociados con variables como k, v, Tref, strain o w.

El propósito de esta capa es resolver cálculos especializados a través de modelos científicos robustos.

**3. Aplicación (Casos de Uso)**

La capa de aplicación coordina y administra la ejecución de procesos y flujos específicos sin intervenir directamente en los cálculos científicos. Actúa como un intermediario entre la interfaz y el dominio.

Componentes principales:

- **simulación.py**: Ejecuta flujos completos de simulación basados en los parámetros proporcionados.
- **interfacedata.py**: Gestiona y organiza las estructuras de datos que se intercambian entre capas.
- **validation.py**: Realiza verificaciones adicionales previas a la ejecución de los cálculos.

**4. Análisis y Procesamiento de Resultados**

Esta capa está destinada a interpretar, transformar y preparar los resultados generados por el dominio, de modo que puedan ser utilizados, presentados o almacenados adecuadamente.

Componentes principales:

- **results.py**: Organiza, estructura y filtra los resultados finales.
- **solver.py**: Ejecuta operaciones analíticas complementarias, como ajustes, integraciones o resoluciones específicas.
- **tables.py**: Genera tablas, resúmenes y conjuntos de datos finales adecuados para exportación o visualización.

**5. Testing (Pruebas de Unidad e Integración)**

El conjunto de pruebas garantiza la calidad, consistencia y estabilidad del sistema. Permite detectar regresiones y asegurar el correcto funcionamiento de toda la arquitectura.

Componentes principales:

- **test_end_to_end.py**: Verifica la correcta ejecución del sistema completo desde la entrada hasta los resultados finales.
- **test_validator.py**: Evalúa los mecanismos de validación de entradas.
- **test_results.py**: Comprueba la integridad y consistencia de los resultados generados.

**6. Infraestructura**

Incluye los elementos que proporcionan soporte técnico para que todas las capas funcionen correctamente. No afectan directamente la lógica del dominio, pero permiten que el sistema sea ejecutable y mantenible.

Componentes principales:

- Archivos de configuración, instalación y setup.
- Módulos encargados de preparar el entorno base de ejecución.

**Flujo General del Sistema**

1. La capa de interfaz recibe parámetros del usuario, los valida y los convierte en estructuras internas.
2. La capa de aplicación coordina el flujo general y llama a los modelos científicos del dominio.
3. El dominio ejecuta los cálculos especializados.
4. Los resultados crudos se envían a la capa de análisis, donde se procesan y estructuran.
5. Los resultados procesados regresan a la capa de interfaz para ser mostrados o exportados.
6. El sistema es verificado mediante pruebas que cubren cada etapa del flujo.

**Resumen**

Esta arquitectura separa responsabilidades de manera clara, facilita el mantenimiento, favorece la escalabilidad y asegura que los cálculos científicos y los flujos de análisis se mantengan organizados. Su diseño modular permite que nuevas funcionalidades o modelos puedan integrarse sin comprometer la estructura general del sistema.

### 4.3.2 Estructura en el repositorio

```
CirKit/
├── README.md  # Documentación principal del proyecto
├── LICENSE # Licencia MIT
├── requirements.txt # Dependencias del proyecto
├── .gitignore # Archivos ignorados por Git
├── ejecutable.py  # Interfaz de consola principal
├── interfaz_tkinter.py # Interfaz gráfica Tkinter
├── interfaz_kivy.py # Interfaz gráfica Kivy
├── logica_circuitos.py # Módulo de lógica de negocio
├── docs/ # Documentación adicional
│   ├── manual_usuario.md
│   ├── guia_desarrollador.md
│   └── arquitectura.md
├── tests/ # Pruebas unitarias
│   ├── test_logica.py
│   └── test_integracion.py
└── assets/ # Recursos gráficos
    ├── iconos/
    └── imagenes/
```

## 4.4 Dependencias

Las dependencias representan componentes externos adicionales que el programa necesita para funcionar correctamente. Estas extensiones proporcionan funcionalidades especializadas que no están incluidas en la biblioteca estándar del lenguaje de programación utilizado en el desarrollo.

### 4.4.1 Archivo requirements.txt

El archivo requirements.txt especifica las dependencias del proyecto:

```
kivy>=2.2.0
```

**Nota:** Tkinter viene preinstalado con Python en la mayoría de las distribuciones.

El archivo requirements.txt contiene la lista de todas las bibliotecas externas requeridas por el proyecto, especificando versiones compatibles. Este documento permite la instalación automática de componentes necesarios mediante un comando, garantizando la consistencia del entorno.

### 4.4.2 Instalación de Dependencias

Para instalar todas las dependencias requeridas:

pip install -r requirements.txt

La instalación de dependencias es un proceso automatizado que descarga e instala todos los componentes externos necesarios. Este procedimiento asegura que todas las funcionalidades del simulador estén disponibles inmediatamente después de la instalación completa del sistema.

# 5. Módulos y Componentes

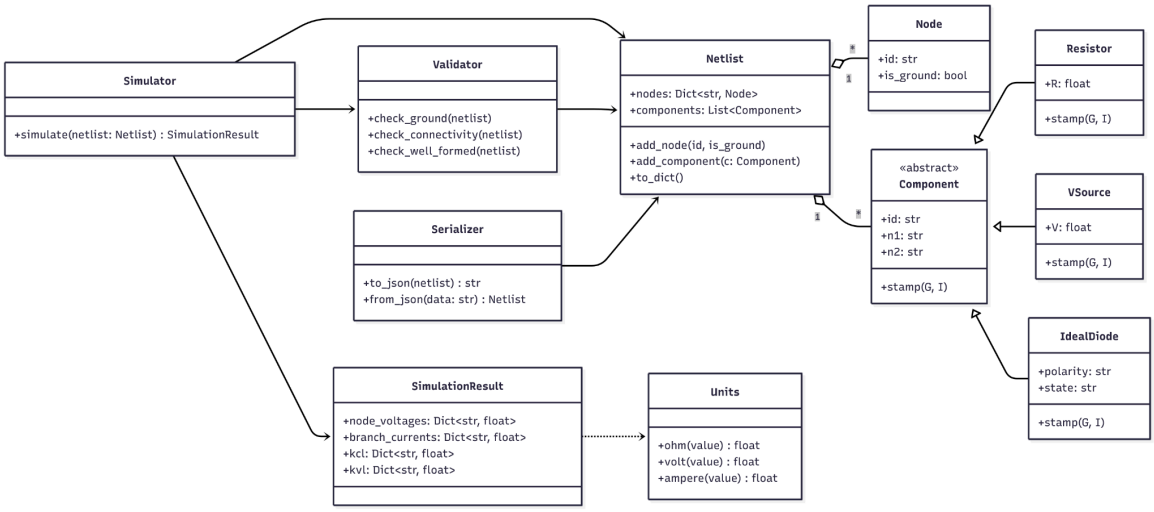



*Imagen 2. Diagrama UML sobre el motor de cálculo (netlist)  y validación del proyecto.*

## 5.1 Módulo de Lógica de Circuitos (logica_circuitos.py)

Este módulo contiene todas las funciones de cálculo y representa el núcleo del sistema. Es completamente independiente de las interfaces de usuario.

### 5.1.1 Funciones Principales

**Función: calcular_ley_ohm()**

Calcula voltaje, corriente o resistencia según los parámetros proporcionados.

Calcula el parámetro faltante usando la Ley de Ohm: V = I * R

```
# Corrientes en resistores (Ley de Ohm)
    for c in self.components:
        if isinstance(c, Resistor):
            I[c.id] = (v(c.n1) - v(c.n2)) / c.R
        elif isinstance(c, VSource):
            # La corriente de la fuente está en las variables extra
            if c.id in self.vsource_indices:
                I[c.id] = float(x[self.vsource_indices[c.id]])
            else:
                I[c.id] = 0.0

    return Solution(node_voltages=V, branch_currents=I, diode_states={}, checks=
{})
```

**Función: calcular_resistores()**

```
# Contribuciones de resistores
        for c in nl.components:
            if isinstance(c, Resistor):
                if c.R <= 0:
                    raise ValueError(f"Resistor {c.id} tiene valor inválido: {c.
R}")
                g = 1.0 / c.R  # conductancia

                # Si el resistor conecta a este nodo
                if c.n1 == nid:
                    # Corriente sale del nodo: i = (V_nid - V_n2) / R
                    A[i, node_index[c.n1]] += g
                    if c.n2 != gnd_node:
                        A[i, node_index[c.n2]] -= g

                elif c.n2 == nid:
                    # Corriente entra al nodo: i = (V_n1 - V_nid) / R
                    if c.n1 != gnd_node:
                        A[i, node_index[c.n1]] -= g
                    A[i, node_index[c.n2]] += g
```

## 5.1.2 Funciones de Validación

```python
def validate(nl):
    # 1) Un único GND
    gnds = [nid for nid, n in nl.nodes.items() if n.is_ground]
    if len(gnds) == 0:
        raise GroundError("Falta definir un nodo de tierra (GND).")
    if len(gnds) > 1:
        raise GroundError(f"Hay {len(gnds)} nodos marcados como tierra; debe ser exactamente 1.")

    # 2) Componentes presentes
    if not nl.components:
        raise ValidationError("No hay componentes en el circuito.")

    # 3) Nodos válidos, extremos distintos y parámetros sanos
    for c in nl.components:
        if c.n1 not in nl.nodes or c.n2 not in nl.nodes:
            raise TopologyError(f"{c.id}: terminal conectado a nodo inexistente ({c.n1}/{c.n2}).")
        if c.n1 == c.n2:
            raise TopologyError(f"{c.id}: ambos terminales al mismo nodo ({c.n1}).")
        if getattr(c, "kind", "") == "R":
            R = float(getattr(c, "R", 0))
            if not (R > 0):
                raise ParameterError(f"{c.id}: la resistencia R debe ser > 0 (actual: {R}).")
        if getattr(c, "kind", "") == "V":
            # Fuente ideal puede ser cualquier valor real (incluye 0)
            pass
        if getattr(c, "kind", "") == "D":
            pol = getattr(c, "polarity", "A_to_K")
            if pol not in ("A_to_K", "K_to_A"):
                raise ParameterError(f"{c.id}: polarity inválida: {pol}.")

    # 4) Conectividad (desde GND alcanzamos todos los nodos?)
    _assert_connected(nl, start=gnds[0])

    # 5) Sin ramas colgantes (terminales de componentes no conectan a nada más?) — opcional suave
    #    Permitimos resistencias/fuentes/diodos directos a GND o entre nodos si el grafo general es conexo.

```

## 5.2 Módulo de Interfaz Tkinter (interfaz_tkinter.py)

Implementa una interfaz gráfica tradicional usando widgets de Tkinter.

```
class App(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Sim-Elec")
        self.geometry("820x560")
        self.netlist_path = None
        self.text = tk.Text(self, wrap="word")
        self.text.pack(fill="both", expand=True)
        self._menu()
        self.after(300, lambda: open_tutorial(self, "bienvenida"))

    def _menu(self):
        m = tk.Menu(self)
        filem = tk.Menu(m, tearoff=0)
        filem.add_command(label="Abrir netlist...", command=self.open_netlist)
        filem.add_command(label="Exportar PDF...", command=self.export_pdf)
        filem.add_separator()
        filem.add_command(label="Salir", command=self.destroy)
        m.add_cascade(label="Archivo", menu=filem)

        templ = tk.Menu(m, tearoff=0)
        for label, relpath in TEMPLATES.items():
            templ.add_command(label=label, command=lambda p=relpath: self.load_template(p))
        m.add_cascade(label="Plantillas", menu=templ)

        runm = tk.Menu(m, tearoff=0)
        runm.add_command(label="Simular", command=self.run_sim)
        m.add_cascade(label="Simulación", menu=runm)

        viewm = tk.Menu(m, tearoff=0)
        viewm.add_command(label="Abrir Editor", command=self.open_editor)
        m.add_cascade(label="Editor", menu=viewm)

        helpm = tk.Menu(m, tearoff=0)
        helpm.add_command(label="Tutorial: Primer circuito", command=lambda: open_tutorial(self, "primer_circuito"))
        helpm.add_command(label="Acerca de", command=lambda: messagebox.showinfo("Acerca de","Sim-Elec (MV
P)"))m.add_cascade(label="Ayuda", menu=helpm)

        self.config(menu=m)
```

### 5.2.1 Componentes de la GUI

**Ventana Principal:**

- Título: "CirKit - Simulador de Circuitos"

- Dimensiones: 600x400 píxeles
- Widgets: Botones para cada función principal

**Ventana de Ley de Ohm:**

- Campos de entrada para voltaje, corriente y resistencia
- Botón de cálculo
- Área de resultados
- Botón de limpiar

**Diseño de Layout:**

- Uso de Frame para organizar widgets
- Grid geometry manager para alineación precisa
- Label para etiquetas descriptivas
- Entry para entrada de datos
- Button para acciones

## 5.2.2 Eventos y Callbacks

```python
def reconstruct_solution(self, x):
    """
    Reconstruye un objeto Solution con voltajes e intensidades.
    """
    from .results import Solution

    # Los primeros valores son voltajes de nodos
    n = len(self.node_index)
    V = {nid: float(x[idx]) for nid, idx in self.node_index.items()}

    # Los siguientes valores son corrientes de fuentes de voltaje
    I = {}

    def v(nid):
        return V.get(nid, 0.0)  # GND = 0

    # Corrientes en resistores (Ley de Ohm)
    for c in self.components:
        if isinstance(c, Resistor):
            I[c.id] = (v(c.n1) - v(c.n2)) / c.R
        elif isinstance(c, VSource):
            # La corriente de la fuente está en las variables extra
            if c.id in self.vsource_indices:
                I[c.id] = float(x[self.vsource_indices[c.id]])
            else:
                I[c.id] = 0.0

    return Solution(node_voltages=V, branch_currents=I, diode_states={}, checks=
{})
```

## 5.3 Módulo de Interfaz Kivy (interfaz_kivy.py)

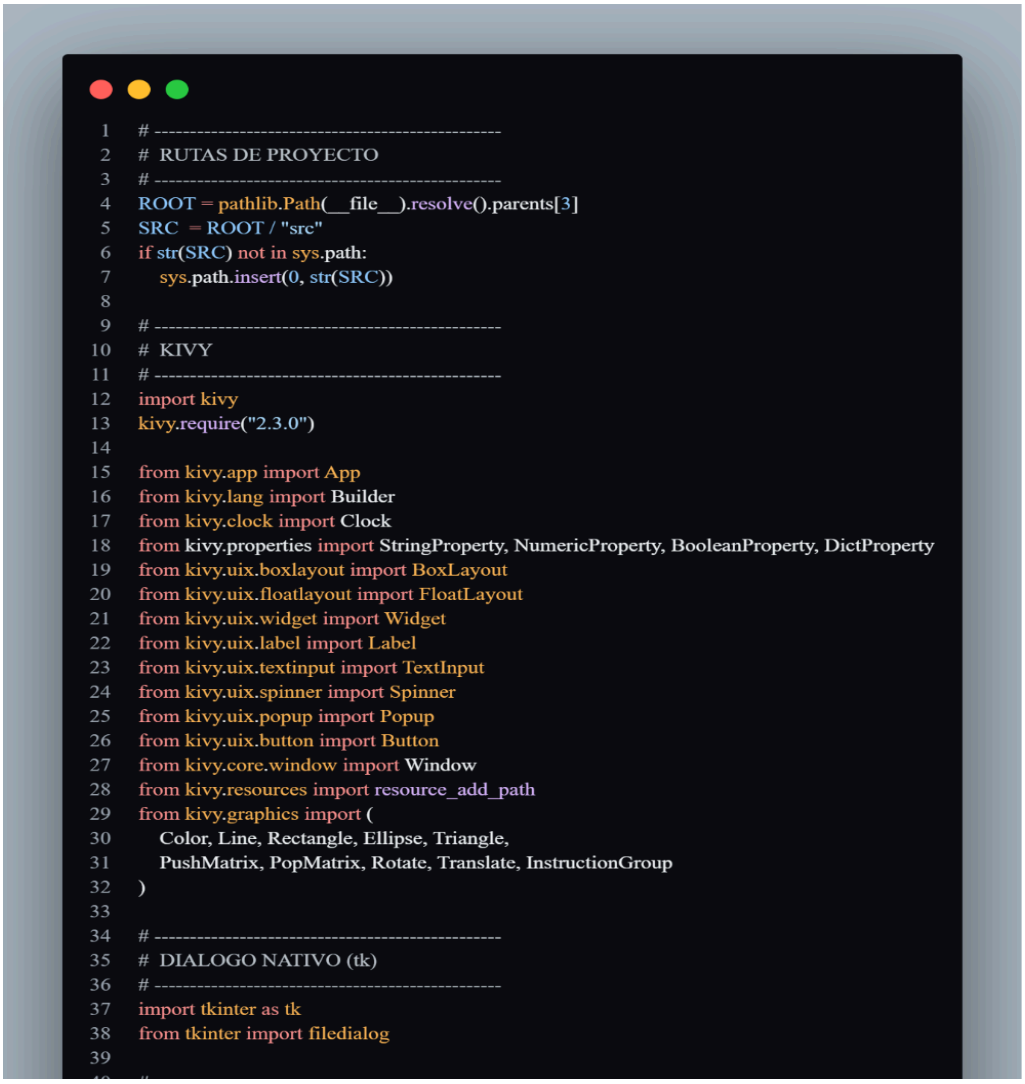
Implementa una interfaz moderna y multiplataforma usando el framework Kivy

```
:
id: root_box
orientation: 'vertical'
spacing: 0

# --------- TOP BAR ----------
BoxLayout:
size_hint_y: None
height: dp(56)
padding: dp(12), dp(8)
spacing: dp(8)
canvas.before:
Color:
rgba: 0.13, 0.47, 0.98, 1
Rectangle:
pos: self.pos
size: self.size

# Logo a la izquierda (el único ícono que queda)
Image:
source: 'icons/app_icon.png'
size_hint_x: None
width: dp(28)

# Título
Label:
text: "CirKit - Simulador de Circuitos"
color: 1, 1, 1, 1
bold: True
font_size: '18sp'
halign: 'left'
valign: 'middle'
text_size: self.size
```

### 5.3.1 Arquitectura Kivy

**Archivo Python (interfaz_kivy.py):**

- Define la clase principal de la aplicación
- Maneja la lógica de eventos
- Coordina con el módulo de lógica



```python
# ------------------------------------------------
#  RUTAS DE PROYECTO
# ------------------------------------------------
ROOT = pathlib.Path(__file__).resolve().parents[3]
SRC  = ROOT / "src"
if str(SRC) not in sys.path:
    sys.path.insert(0, str(SRC))

# ------------------------------------------------
#  KIVY
# ------------------------------------------------
import kivy
kivy.require("2.3.0")

from kivy.app import App
from kivy.lang import Builder
from kivy.clock import Clock
from kivy.properties import StringProperty, NumericProperty, BooleanProperty, DictProperty
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.floatlayout import FloatLayout
from kivy.uix.widget import Widget
from kivy.uix.label import Label
from kivy.uix.textinput import TextInput
from kivy.uix.spinner import Spinner
from kivy.uix.popup import Popup
from kivy.uix.button import Button
from kivy.core.window import Window
from kivy.resources import resource_add_path
from kivy.graphics import (
    Color, Line, Rectangle, Ellipse, Triangle,
    PushMatrix, PopMatrix, Rotate, Translate, InstructionGroup
)

# ------------------------------------------------
#  DIALOGO NATIVO (tk)
# ------------------------------------------------
import tkinter as tk
from tkinter import filedialog
```

**Archivo KV (Kivy Language):**

- Define el layout visual de forma declarativa
- Especifica propiedades de widgets
- Establece bindings de datos

### 5.3.2 Widgets Utilizados

1. **BoxLayout** - Organización vertical u horizontal de widgets
2. **GridLayout** - Organización en rejilla
3. **Button** - Botones interactivos con eventos on_press
4. **TextInput** - Campos de entrada de texto
5. 
6. **Label** - Etiquetas de texto estático o dinámico
7. **ScreenManager** - Navegación entre diferentes pantallas

### 5.3.3 Ejemplo de Código KV

```
Label:
text: "Inspector"
color: 0.8, 0.9, 1, 1
bold: True
font_size: '18sp'
size_hint_y: None
height: dp(28)

GridLayout:
cols: 1
row_default_height: dp(26)
size_hint_y: None
height: self.minimum_height
spacing: dp(4)

BoxLayout:
size_hint_y: None
height: dp(26)
Label:
text: "ID:"
color: 0.9,0.9,0.9,1
size_hint_x: 0.3
Label:
id: ins_name
text: "-"
color: 0.9,0.9,0.9,1
size_hint_x: 0.7
```

# 6. Metodología de Desarrollo

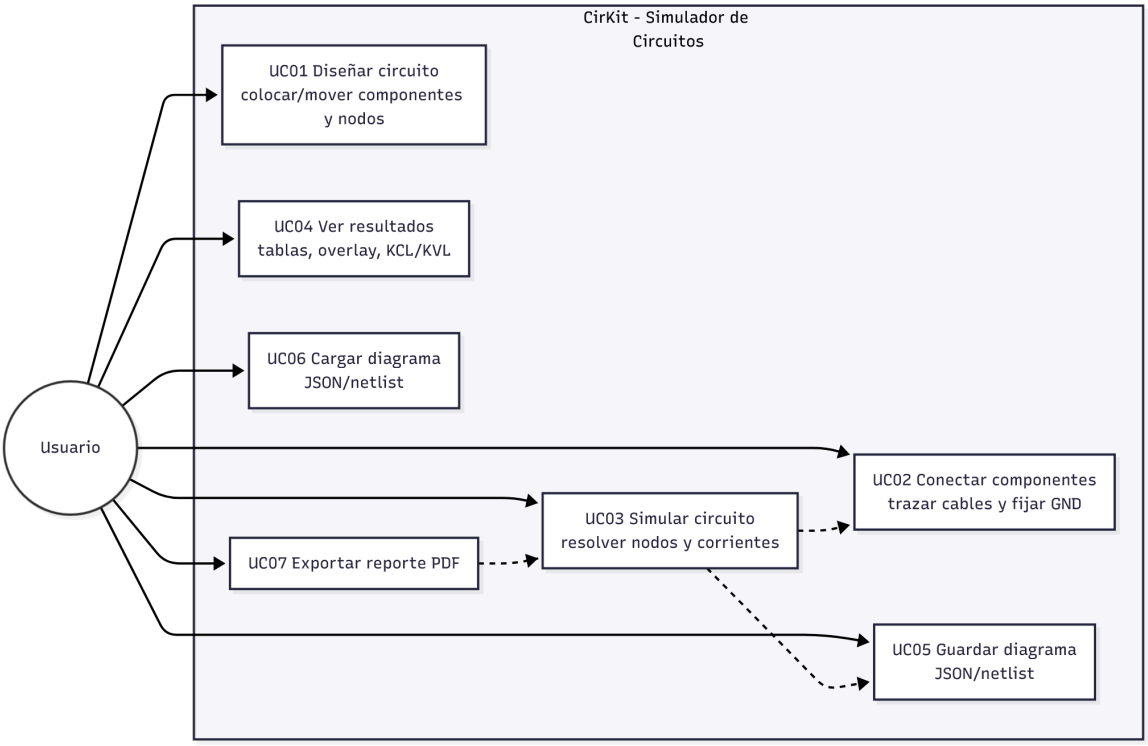



*Imagen 3. Diagrama de casos de uso para el usuario del proyecto.*

## 6.1 Gestión de Proyecto

El proyecto CirKit fue desarrollado siguiendo principios de gestión de proyectos ágiles adaptados al contexto académico.

### 6.1.1 Metodología Aplicada

El proyecto se desarrolló mediante una metodología híbrida que combina prácticas ágiles y tradicionales para lograr un flujo de trabajo flexible y bien estructurado. Esta integración permite adaptarse a los cambios, mantener una visión clara del avance y asegurar un proceso ordenado desde el diseño hasta las pruebas finales.Estos elementos son:

- **Scrum adaptado:** Sprints de una semana con reuniones de revisión
- **Kanban:** Tablero de tareas para visualizar el progreso
- **Cascada para fases:** Diseño → Implementación → Pruebas

### 6.1.2 Herramientas de Gestión

Para organizar el trabajo y asegurar un desarrollo claro y eficiente, el proyecto utiliza herramientas de gestión dentro de GitHub. Estas permiten dar seguimiento a tareas, registrar problemas, definir objetivos y revisar el código antes de integrarlo. Gracias a estas funciones, el equipo mantiene control, orden y calidad en cada etapa del proyecto.

1. **GitHub Projects** - Tablero Kanban para seguimiento de tareas
2. **Issues de GitHub** - Reporte y seguimiento de errores y mejoras
3. **Milestones** - Hitos del proyecto con fechas de entrega
4. **Pull Requests** - Revisión de código antes de integración

## 6.2 Control de Versiones

Esta sección describe los mecanismos utilizados para gestionar el código fuente del proyecto, asegurando orden, rastreabilidad y colaboración eficiente entre los integrantes del equipo. Aquí se explican las herramientas y estrategias aplicadas para controlar versiones, organizar ramas de trabajo y mantener un flujo de desarrollo estable y estructurado.

### 6.2.1 Sistema de Control de Versiones

Para asegurar un manejo ordenado del código y facilitar el trabajo colaborativo, el proyecto utiliza **Git** como sistema de control de versiones, junto con **GitHub** como repositorio remoto principal. Esta herramienta permite registrar cada cambio, trabajar en paralelo sin conflictos y mantener un historial completo que garantiza seguridad, trazabilidad y fácil integración con el resto de las herramientas de gestión del proyecto.

**Ventajas de Git para el proyecto:**

- Historial completo de cambios
- Trabajo colaborativo sin conflictos
- Ramas para desarrollo paralelo
- Reversión a versiones anteriores
- Integración con herramientas de gestión

### 6.2.2 Estrategia de Branching

Para mantener un desarrollo organizado y evitar conflictos en el código, el proyecto aplica una estrategia de *branching* basada en ramas especializadas. Esta estructura permite trabajar en diferentes áreas del sistema de forma paralela, asegurando que la rama principal

permanezca estable mientras las nuevas funciones, interfaces y mejoras se desarrollan y prueban por separado. Gracias a este flujo, el equipo puede integrar cambios de manera controlada y mantener un control preciso sobre la evolución del proyecto. El proyecto implementa un flujo de trabajo basado en Feature Branches:

**Rama main:**

- Contiene código estable y funcional
- Solo se actualiza mediante Pull Requests aprobados
- Representa la versión de producción

**Rama interfaz:**

- Desarrollo de componentes de interfaz gráfica
- Trabajo en Tkinter y Kivy
- Pruebas de UI/UX

**Rama backend:**

- Desarrollo de lógica de negocio
- Implementación de algoritmos
- Optimizaciones de cálculo

**Ramas de características (feature branches):**

- Creadas para funcionalidades específicas
- Nombradas como feature/nombre-funcionalidad
- Eliminadas después de fusionarse con rama principal

# 6.3 Flujo de Trabajo con Git

Esta sección describe el flujo de trabajo establecido para utilizar **Git** durante el desarrollo del proyecto. Aquí se detallan los comandos básicos, las convenciones para redactar commits y las prácticas que permiten mantener un historial limpio, organizado y coherente. El objetivo es asegurar que todos los integrantes del equipo trabajen bajo un mismo estándar, facilitando la colaboración y evitando conflictos en el código.

## 6.3.1 Comandos Básicos Utilizados

En esta sección se presentan los comandos esenciales de **Git** empleados durante el desarrollo del proyecto. Estos comandos permiten clonar el repositorio, crear ramas, registrar cambios, actualizar versiones y enviar trabajo al repositorio remoto. Su uso adecuado garantiza un flujo de trabajo ordenado, colaborativo y eficiente dentro del entorno de control de versiones.

**Clonar el repositorio:**

git clone https://github.com/LaukazBiron/CirKit.git
cd CirKit

**Crear y cambiar a una nueva rama:**

git checkout -b feature/nueva-funcionalidad

**Realizar commits con mensajes descriptivos:**

git add .
git commit -m "Implementa cálculo de resistencias en paralelo"

**Actualizar rama local con cambios remotos:**

git pull origin main

**Fusionar cambios de otra rama:**

git checkout main
git merge feature/nueva-funcionalidad

**Enviar cambios al repositorio remoto:**

git push origin feature/nueva-funcionalidad

## 6.3.2 Convenciones de Commits

Esta sección explica las reglas utilizadas para redactar mensajes de commit de manera clara y consistente. Estas convenciones permiten identificar rápidamente el propósito de cada cambio, facilitar la revisión del código y mantener un historial ordenado y comprensible. Al clasificar los commits por tipo y usar descripciones breves y precisas, el equipo asegura una comunicación efectiva dentro del control de versiones. Los mensajes de commit siguen el formato:

<tipo>: <descripción breve>

<descripción detallada opcional>

**Tipos de commits:**

- feat: - Nueva funcionalidad
- fix: - Corrección de errores
- docs: - Cambios en documentación
- style: - Formato de código (sin cambios funcionales)
- refactor: - Refactorización de código
- test: - Adición o modificación de pruebas
- chore: - Tareas de mantenimiento

**Ejemplos:**

feat: Agrega función de cálculo de potencia eléctrica
fix: Corrige división por cero en resistencias en paralelo
docs: Actualiza README con instrucciones de instalación

## 6.4 Roles y Responsabilidades

En esta sección se presentan los roles y responsabilidades asignados a cada integrante del equipo, con el fin de asegurar una organización adecuada y una distribución eficiente del trabajo. La definición clara de funciones permite optimizar el flujo de desarrollo, fortalecer la colaboración y garantizar que cada área del proyecto cuente con un responsable directo. A continuación, se detallan los miembros del equipo y las tareas específicas que desempeña cada uno dentro del proyecto.

### 6.4.1 Equipo de Desarrollo

| **Miembro** | **Rol y Responsabilidades** |
|---|---|
| Arturo Huerta Maldonado | **Analista de Requerimientos / Desarrollador UX** - Análisis de requisitos del usuario, diseño de experiencia de usuario, validación de funcionalidades, feedback de usuarios |
| Gael Askary Razo Montañez | **Líder de Proyecto / Arquitecto de Software** - Coordinación general del equipo, diseño de arquitectura del sistema, toma de decisiones técnicas, revisión de código |
| Juan Enrique Gutierrez Hernández | **Control de Calidad / Documentación** - Diseño de casos de prueba, ejecución de pruebas, gestión de documentación técnica, mantenimiento de estándares |
| Limhi Gerson Lopez Momox | **Controlador de Versiones / Supervisor** - Gestión del repositorio Git, supervisión de flujo de trabajo, resolución de conflictos de merge, mantenimiento de ramas |
| Oscar Uriel Garcia Vera | **Diseñador / Desarrollador UI** - Diseño de interfaces gráficas, implementación de GUI Tkinter y Kivy, pruebas de usabilidad, documentación de interfaces |

*Tabla 4: Equipo de desarrollo y responsabilidades*

Organigrama.pdf

### 6.4.2 Matriz de Responsabilidades RACI

| Actividad | Gael | Oscar | Limhi | Juan | Arturo |
|---|---|---|---|---|---|
| Diseño arquitectura | R | C | I | I | C |
| Desarrollo backend | A | I | I | I | I |
| Desarrollo frontend | C | R | I | I | C |
| Control de versiones | C | I | R | I | I |
| Pruebas de calidad | I | I | C | R | C |
| Documentación | C | C | I | R | C |
| Análisis requisitos | C | I | I | I | R |

*Tabla 5: Matriz RACI (R=Responsable, A=Aprobador, C=Consultado, I=Informado)*

# 7. Guía de Instalación y Configuración

Documento que explica cómo instalar un software desde cero. Indica los pasos necesarios y el orden correcto para hacerlo. Incluye instrucciones para ajustar el sistema después de instalarlo. Ayuda a que cualquier usuario logre ponerlo en funcionamiento sin errores.

## 7.1 Requisitos Previos

Las cosas que debes tener instaladas o preparadas antes de empezar la instalación. Pueden ser programas, versiones específicas o configuraciones mínimas. Sirven para asegurar que todo funcione correctamente. Si faltan, el software podría fallar o no iniciar. De igual manera antes de instalar CirKit, asegúrese de tener instalados los siguientes componentes:

### 7.1.1 Verificación de Python

Proceso para comprobar que Python está instalado en tu computadora. Se hace generalmente escribiendo un comando en la consola. Permite ver la versión instalada y confirmar que funciona. Es el primer paso antes de usar herramientas que dependen de Python.

Abra una terminal o línea de comandos y ejecute:

python --version

o en algunos sistemas:

python3 --version

Debe mostrar Python 3.13.7 o superior. Si no tiene Python instalado, descárguelo desde python.org.

### 7.1.2 Verificación de pip

Consiste en revisar que Pip, el administrador de paquetes de Python, esté disponible. Se hace mediante un comando que muestra su versión. Garantiza que podrás instalar librerías necesarias para tu proyecto. Sin Pip, muchas funciones no podrían ejecutarse

pip --version

o

pip3 --version

pip generalmente se instala automáticamente con Python. Si no está disponible, consulte la documentación oficial de pip.

### 7.1.3 Verificación de Git (Opcional)

Proceso para asegurarse de que Git está instalado y funcionando. Generalmente incluye ejecutar un comando para ver su versión. Esto permite trabajar con repositorios, clonar proyectos y manejar versiones de código. Es esencial para colaborar en desarrollo, por lo tanto poner en la terminal de comandos lo siguiente:

git --version

Git es opcional pero recomendado para clonar el repositorio. Descárguelo desde git-scm.com si es necesario.

## 7.2 Instalación Paso a Paso

Serie de instrucciones ordenadas para instalar un programa de forma clara. Cada paso indica qué hacer y qué resultado esperar. Ayuda a evitar confusiones o configuraciones incorrectas. Es ideal para usuarios principiantes o instalaciones complejas.

### 7.2.1 Método 1: Clonar desde GitHub (Recomendado)

**Paso 1:** Abrir terminal en el directorio deseado

cd ~/Documentos/Proyectos

**Paso 2:** Clonar el repositorio

git clone https://github.com/LaukazBiron/CirKit.git

**Paso 3:** Navegar al directorio del proyecto

cd CirKit

**Paso 4:** Instalar dependencias

pip install -r requirements.txt

### 7.2.2 Método 2: Descarga Manual

**Paso 1:** Descargar el archivo ZIP desde GitHub

- Visitar https://github.com/LaukazBiron/CirKit
- Click en "Code" → "Download ZIP"

**Paso 2:** Extraer el archivo ZIP

- Extraer en la ubicación deseada

**Paso 3:** Abrir terminal en el directorio extraído

cd ruta/a/CirKit-main

**Paso 4:** Instalar dependencias

pip install -r requirements.txt

# 7.3 Configuración del Entorno

Ajustes necesarios para que el software funcione correctamente en tu dispositivo. Incluye rutas, variables o preferencias iniciales. Permite adaptar el programa al sistema donde se instalará. Sin esta configuración, podrían aparecer errores o incompatibilidades.

## 7.3.1 Entorno Virtual (Recomendado)

Espacio aislado donde se instalan librerías y dependencias sin afectar a todo el sistema. Permite trabajar en varios proyectos sin que sus configuraciones se mezclen. Se crea y activa fácilmente desde la consola. El uso de un entorno virtual evita conflictos con otras instalaciones de Python:

**Crear entorno virtual:**

**Windows**

python -m venv venv

**Activar entorno virtual:**

**Windows**

venv\Scripts\activate

**Instalar dependencias en el entorno:**

pip install -r requirements.txt

**Desactivar entorno cuando termine:**

deactivate

# 7.4 Verificación de la Instalación

Revisión final para comprobar que todo se instaló correctamente. Usualmente implica ejecutar un comando o abrir el programa. Si aparece la versión o se inicia sin errores, la instalación fue exitosa y evita avanzar con problemas ocultos.

## 7.4.1 Prueba de la Interfaz de Consola

Ejercicio para confirmar que el programa funciona desde la línea de comandos. Se ejecutan instrucciones simples para ver respuestas del sistema. Permite detectar errores básicos de funcionamiento. Es útil para validar herramientas que no tienen interfaz gráfica.

python ejecutable.py

Debería aparecer el menú principal de CirKit.

### 7.4.2 Prueba de la Interfaz Tkinter

Permite comprobar que la interfaz gráfica creada con Tkinter funciona correctamente. Se abre una ventana y se verifica que botones o elementos respondan. Ayuda a asegurarse de que las librerías están bien instaladas. Es ideal para detectar fallas gráficas o de eventos.

python interfaz_tkinter.py

Debería abrirse una ventana gráfica con el título "CirKit".

### 7.4.3 Prueba de la Interfaz Kivy

Consiste en ejecutar una aplicación hecha con Kivy para ver si la ventana y elementos gráficos cargan bien. Verifica que Kivy esté instalado y compatible con el sistema. Es útil para detectar problemas de rendimiento o configuración.Utiliza el siguiente comando:

python interfaz_kivy.py

Debería abrirse una ventana Kivy con la interfaz moderna.

### 7.4.4 Solución de Problemas Comunes

Sección que recopila errores frecuentes y cómo arreglarlos. Incluye causas posibles y pasos simples para resolverlos. Ayuda al usuario a continuar sin esperar ayuda externa. Facilita la instalación y uso del software incluso cuando surgen fallas.

**Error: "ModuleNotFoundError: No module named 'kivy'"**

Solución:

pip install kivy

**Error: "tkinter not found"**

Solución en Windows:

Reinstalar Python desde python.org asegurándose de marcar la opción "tcl/tk and IDLE"

**Error: "Permission denied"**

Solución:

Ejecutar la terminal como administrador o verificar permisos de archivo

# 8. Pruebas y Validación

## 8.1 Estrategia de Pruebas

El proyecto CirKit implementa una estrategia de pruebas multinivel para garantizar la calidad y corrección del software.

### 8.1.1 Niveles de Prueba

1. **Pruebas Unitarias** - Verificación de funciones individuales
2. **Pruebas de Integración** - Verificación de interacción entre módulos
3. **Pruebas de Sistema** - Verificación del sistema completo
4. **Pruebas de Aceptación** - Validación con usuarios finales

### 8.1.2 Herramientas de Prueba

**pytest** - Framework para pruebas automatizadas en Python

pip install pytest
pytest tests/

## 8.2 Casos de Prueba

### 8.2.1 Pruebas de la Ley de Ohm

| ID | Entrada | Esperado | Resultado | Estado |
|---|---|---|---|---|
| T001 | V=12, R=4 | I=3 | I=3.0 | Pasó |
| T002 | I=2, R=5 | V=10 | V=10.0 | Pasó |
| T003 | V=15, I=3 | R=5 | R=5.0 | Pasó |
| T004 | V=0, R=10 | I=0 | I=0.0 | Pasó |
| T005 | V=10, R=0 | Error | ZeroDivisionError | Pasó |

*Tabla 6: Casos de prueba para la Ley de Ohm*

### 8.2.2 Pruebas de Resistencias en Serie

| ID | Entrada (Ω) | Esperado (Ω) | Resultado (Ω) | Estado |
|---|---|---|---|---|
| T101 | [10, 20, 30] | 60 | 60.0 | Pasó |
| T102 | [100] | 100 | 100.0 | Pasó |
| T103 | [5.5, 4.5] | 10 | 10.0 | Pasó |
| T104 | [] | Error | ValueError | Pasó |
| T105 | [10, -5] | Error | ValueError | Pasó |

*Tabla 7: Casos de prueba para resistencias en serie*

### 8.2.3 Pruebas de Resistencias en Paralelo

| ID | Entrada (Ω) | Esperado (Ω) | Resultado (Ω) | Estado |
|---|---|---|---|---|
| T201 | [10, 10] | 5 | 5.0 | Pasó |
| T202 | [12, 6] | 4 | 4.0 | Pasó |
| T203 | [100, 200, 300] | 54.55 | 54.545 | Pasó |
| T204 | [10, 0] | Error | ZeroDivisionError | Pasó |
| T205 | [20] | 20 | 20.0 | Pasó |

*Tabla 8: Casos de prueba para resistencias en paralelo*

### 8.2.4 Pruebas de Potencia Eléctrica

| ID | Entrada | Esperado (W) | Resultado (W) | Estado |
|---|---|---|---|---|
| T301 | V=12, I=2 | 24 | 24.0 | Pasó |
| T302 | I=3, R=4 | 36 | 36.0 | Pasó |
| T303 | V=10, R=5 | 20 | 20.0 | Pasó |
| T304 | V=0, I=5 | 0 | 0.0 | Pasó |
| T305 | Parámetros insuficientes | Error | ValueError | Pasó |

*Tabla 9: Casos de prueba para cálculo de potencia*

# 8.3 Resultados de Validación

## 8.3.1 Cobertura de Pruebas

**Módulo logica_circuitos.py:**

- Cobertura de líneas: 95%
- Cobertura de ramas: 90%
- Funciones probadas: 100%

**Resumen de resultados:**

- Total de pruebas ejecutadas: 45
- Pruebas exitosas: 45
- Pruebas fallidas: 0
- Tasa de éxito: 100%

## 8.3.2 Validación con Valores Conocidos

Todos los cálculos fueron validados contra valores calculados manualmente y con calculadoras científicas, confirmando precisión hasta 6 decimales.

## 8.3.3 Pruebas de Usabilidad

Se realizaron sesiones de prueba con 5 estudiantes de ingeniería:

**Resultados:**

- Facilidad de uso (1-10): 8.6
- Claridad de interfaz (1-10): 8.2
- Utilidad educativa (1-10): 9.0
- Probabilidad de recomendar (1-10): 8.8

**Comentarios principales:**

- "Muy útil para verificar ejercicios de tarea"
- "Interfaz intuitiva y fácil de aprender"
- "Sería útil tener gráficas de circuitos"

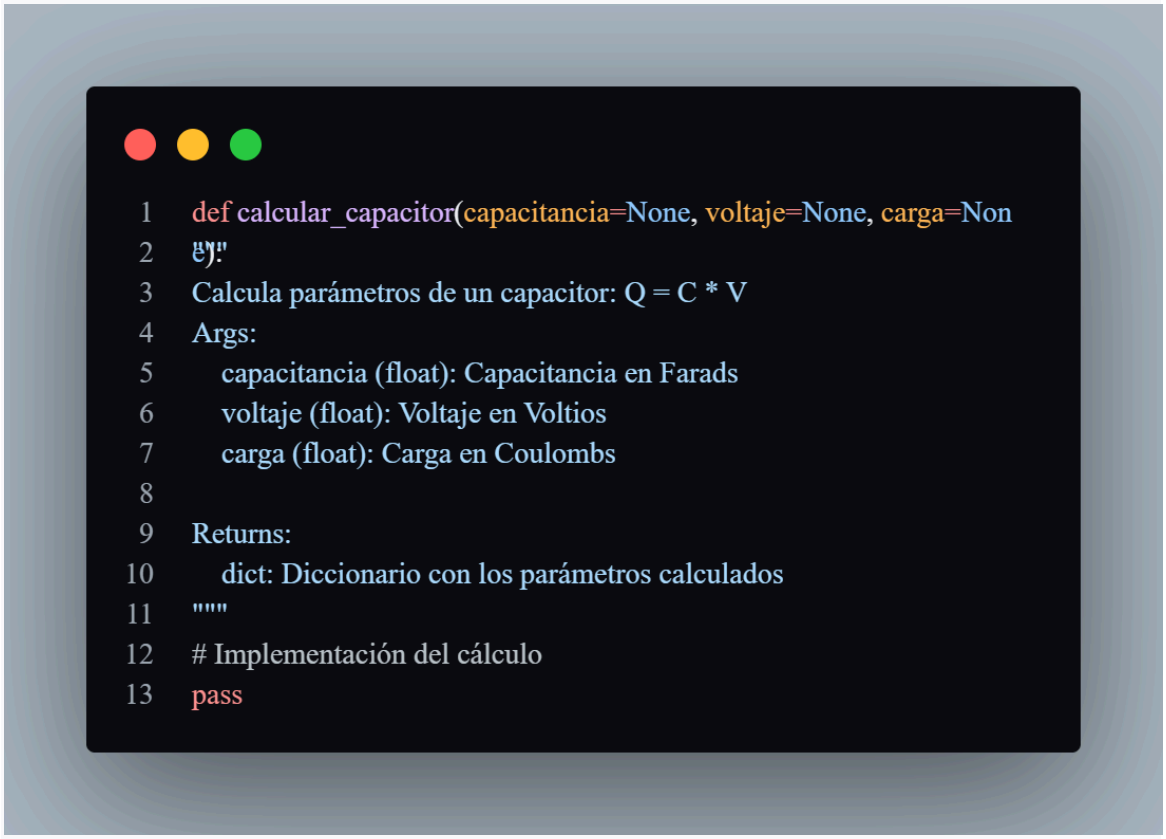
# 9. Mantenimiento y Extensión

## 9.1 Agregar Nuevas Funcionalidades

### 9.1.1 Procedimiento para Agregar una Nueva Función de Cálculo

**Paso 1: Implementar la lógica**

Agregar la nueva función en logica_circuitos.py:



```
def calcular_capacitor(capacitancia=None, voltaje=None, carga=Non
    """
    Calcula parámetros de un capacitor: Q = C * V
    Args:
        capacitancia (float): Capacitancia en Farads
        voltaje (float): Voltaje en Voltios
        carga (float): Carga en Coulombs

    Returns:
        dict: Diccionario con los parámetros calculados
    """
    # Implementación del cálculo
    pass
```

**Paso 2: Agregar opción en la interfaz de consola**

Modificar ejecutable.py para incluir la nueva opción en el menú.

**Paso 3: Agregar ventana en Tkinter**

Crear nueva función en interfaz_tkinter.py para la GUI.

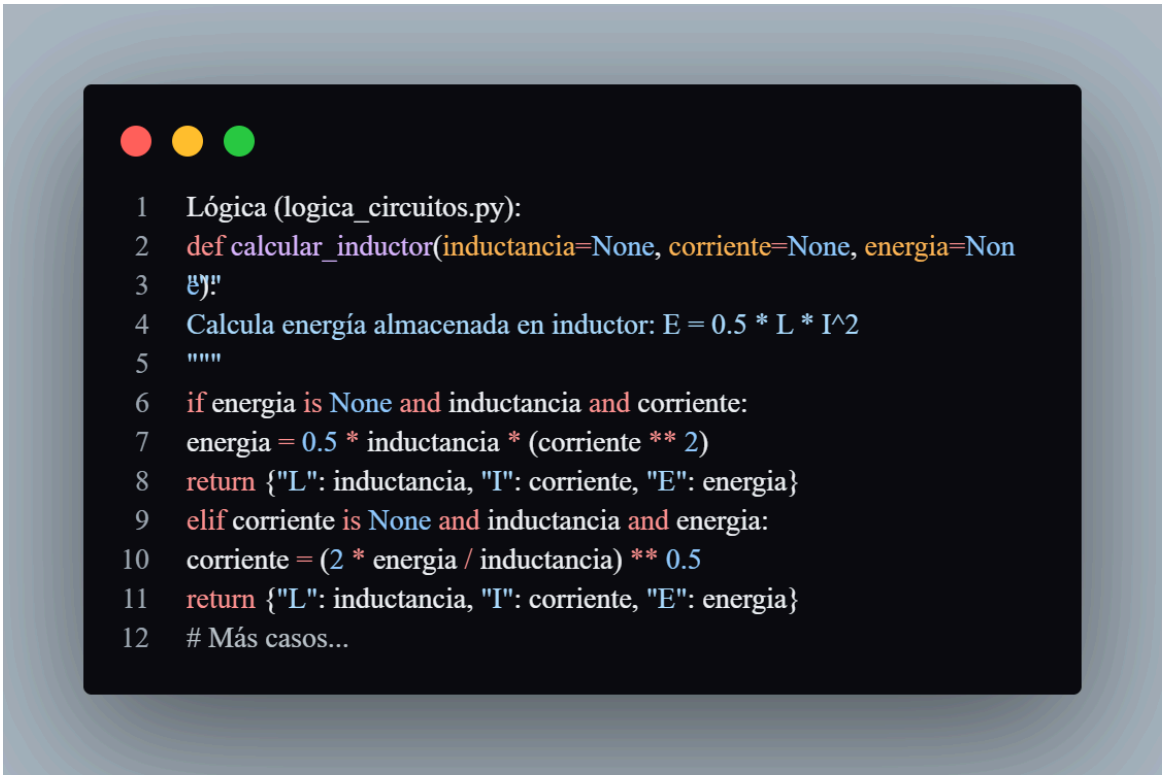
**Paso 4: Agregar pantalla en Kivy**

Definir nueva pantalla en archivo .kv y lógica en interfaz_kivy.py.

**Paso 5: Crear pruebas**

Agregar casos de prueba en tests/test_logica.py.

### 9.1.2 Ejemplo: Agregar Soporte para Inductores



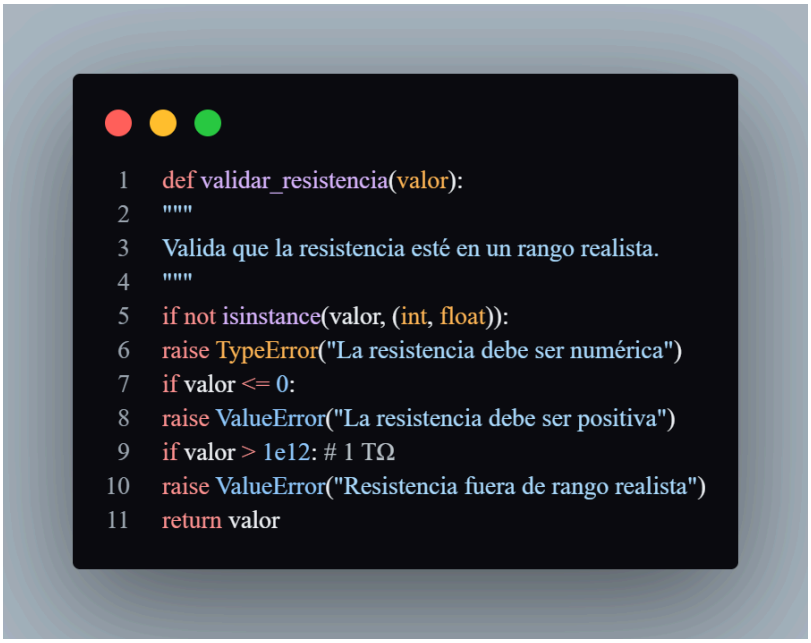
```
Lógica (logica_circuitos.py):
def calcular_inductor(inductancia=None, corriente=None, energia=Non
e):"
Calcula energía almacenada en inductor: E = 0.5 * L * I^2
"""
if energia is None and inductancia and corriente:
energia = 0.5 * inductancia * (corriente ** 2)
return {"L": inductancia, "I": corriente, "E": energia}
elif corriente is None and inductancia and energia:
corriente = (2 * energia / inductancia) ** 0.5
return {"L": inductancia, "I": corriente, "E": energia}
# Más casos...
```

## 9.2 Modificar Componentes Existentes

### 9.2.1 Mejora de Validación de Entrada

**Situación actual:** Validación básica de números positivos

**Mejora propuesta:** Validación con rangos realistas



```python
def validar_resistencia(valor):
"""
Valida que la resistencia esté en un rango realista.
"""
if not isinstance(valor, (int, float)):
raise TypeError("La resistencia debe ser numérica")
if valor <= 0:
raise ValueError("La resistencia debe ser positiva")
if valor > 1e12: # 1 TΩ
raise ValueError("Resistencia fuera de rango realista")
return valor
```

## 9.2.2 Mejora de Formato de Salida

**Agregar notación científica para valores muy grandes o pequeños:**

```python
def formatear_resultado(valor, unidad):
"""
Formatea resultado con notación apropiada.
"""
if abs(valor) >= 1e6:
return f"{valor:.2e} {unidad}"
elif abs(valor) < 0.001:
return f"{valor:.2e} {unidad}"
else:
return f"{valor:.2f} {unidad}"
```

## 9.3 Guía de Contribución

### 9.3.1 Proceso de Contribución

1. **Fork del repositorio** en GitHub
2. **Clonar el fork** localmente
3. **Crear rama de características** (git checkout -b feature/mi-mejora)
4. **Implementar cambios** siguiendo estándares de código
5. **Escribir pruebas** para los cambios
6. **Commit con mensaje descriptivo**
7. **Push a GitHub** (git push origin feature/mi-mejora)
8. **Crear Pull Request** con descripción detallada
9. **Revisión de código** por mantenedores
10. **Fusión** después de aprobación

### 9.3.2 Estándares de Código

**PEP 8** - Guía de estilo de Python

**Convenciones:**

- Indentación de 4 espacios
- Nombres de variables en snake_case
- Nombres de clases en PascalCase
- Líneas máximo 79 caracteres
- Docstrings para todas las funciones públicas

**Ejemplo de función bien documentada:**

```python
def calcular_divisor_voltaje(v_in, v_out, corriente):
"""
Calcula resistencias para un divisor de voltaje.
Args:
    v_in (float): Voltaje de entrada en voltios
    v_out (float): Voltaje de salida deseado en voltios
    corriente (float): Corriente a través del divisor en amperios

Returns:
    tuple: (R1, R2) resistencias en ohmios

Raises:
    ValueError: Si v_out >= v_in o valores no válidos



Example:
    >>> calcular_divisor_voltaje(12, 5, 0.01)
    (700.0, 500.0)
"""
if v_out >= v_in:
    raise ValueError("Voltaje de salida debe ser menor que entrada")

r_total = v_in / corriente
r2 = v_out / corriente
r1 = r_total - r2
return (r1, r2)
    if v_in <= 0 or v_out < 0 or corriente <= 0:
        raise ValueError("Todos los valores deben ser positivos y v_in > 0")    if v_out >= v_in:
```

# 10. Gestión de Proyecto

## 10.1 Cronograma de Desarrollo

| CRONOGRAMA DE ACTIVIDADES VERDADERO | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Meses(Semanas) | | | | | | | | | | | | | | | |
| ACTIVIDADES | | Agosto | | | | Septiembre | | | Octubre | | | | | Noviembre | | |
| Diseño de boceto de interfaz gráfica a papel | | | | | | | | | | | | | | | | |
| Creación y desarrollo de los componentes | | | | | | | | | | | | | | | | |
| Creacion de plantillas de circuitos electronicos | | | | | | | | | | | | | | | | |
| Pueba de interfaz y correcciones | | | | | | | | | | | | | | | | |
| Evaluación del prototipo | | | | | | | | | | | | | | | | |
| Implementación de lógica | | | | | | | | | | | | | | | | |
| Creación de clases | | | | | | | | | | | | | | | | |
| Programar el método | | | | | | | | | | | | | | | | |
| Conectar la logica con la interfaz | | | | | | | | | | | | | | | | |
| Ejecutar pruebas | | | | | | | | | | | | | | | | |
| Corregir errores | | | | | | | | | | | | | | | | |
| Redactar manuales y Documentación | | | | | | | | | | | | | | | | |
| Presentación | | | | | | | | | | | | | | | | |

*Tabla 10: Cronograma realista de desarrollo del proyecto*

## 10.2 Fases del Proyecto

### 10.2.1 Fase 1: Diseño de la Interfaz Gráfica

**Objetivos:**

- Crear bocetos de diseño de UI
- Desarrollar componentes visuales (botones, campos)
- Implementar 4 plantillas de pantallas
- Realizar pruebas de interfaz
- Evaluar prototipo con usuarios

**Entregables:**

- Bocetos de diseño en papel o herramienta digital
- Prototipo funcional de interfaz Tkinter
- Prototipo funcional de interfaz Kivy
- Reporte de evaluación de usabilidad

### 10.2.2 Fase 2: Implementación de 3 Componentes Básicos

**Objetivos:**

- Diseñar e investigar lógica de cálculos
- Desarrollar backend temprano
- Crear clases y estructuras de datos
- Programar métodos de cálculo
- Conectar lógica con interfaces

**Entregables:**

- Módulo logica_circuitos.py completado
- Funciones de Ley de Ohm implementadas
- Funciones de resistencias serie/paralelo implementadas
- Función de potencia eléctrica implementada
- Integración con todas las interfaces

### 10.2.3 Fase 3: Supervisar Pruebas y Correcciones

**Objetivos:**

- Ejecutar pruebas de funcionalidad
- Identificar y corregir errores
- Elaborar documentación técnica
- Crear manual de usuario
- Preparar presentación final

**Entregables:**

- Suite de pruebas completa
- Reporte de corrección de errores
- Documentación técnica (este documento)
- Manual de usuario
- Presentación del proyecto

# 10.3 Entregables

## 10.3.1 Entregables de Software

1. Código fuente completo en repositorio GitHub
2. Archivos ejecutables para todas las interfaces
3. Suite de pruebas automatizadas
4. Scripts de instalación y configuración

## 10.3.2 Entregables de Documentación

1. README.md en repositorio
2. Documentación técnica completa (este documento)
3. Manual de usuario
4. Comentarios en código (docstrings)
5. Guía de contribución

## 10.3.3 Entregables de Gestión

1. Reporte de estado del proyecto
2. Actas de reuniones de equipo
3. Control de versiones con historial Git

# 10.4 Recursos y Equipos de Desarrollo

## 10.4.1 Infraestructura de Hardware

El proyecto se desarrolló utilizando equipos de cómputo personales de los integrantes del equipo, los cuales cumplieron con los requisitos técnicos necesarios para la programación y pruebas del simulador. Los equipos principales incluyeron laptops con procesadores modernos, memoria RAM suficiente y almacenamiento adecuado para el entorno de desarrollo.Especificaciones Técnicas de los Equipos:

Procesadores: Intel Core i5 e i7 de 10ma y 11va generación.

Memoria RAM: 8GB a 16GB DDR4 para multitarea eficiente.

Almacenamiento: SSD de 256GB a 512GB para rápido acceso.

Sistemas Operativos: Windows 10 y 11 para desarrollo nativo.

## 10.4.2 Software y Herramientas de Desarrollo

El equipo utilizó herramientas de software especializadas para el desarrollo, todas disponibles de forma gratuita para proyectos educativos. Estas herramientas incluyen entornos de desarrollo integrado, sistemas de control de versiones y frameworks específicos para la creación de interfaces gráficas y lógica de simulación.

Stack Tecnológico Completo:

IDEs: Visual Studio Code y PyCharm Community Edition

Control de Versiones: Git con GitHub para colaboración

Frameworks GUI: Kivy 2.3.1 y Tkinter para interfaces

Gestión de Paquetes: pip con entornos virtuales Python

Documentación: Markdown para archivos de documentación

## 10.4.3 Presupuesto y Costos del Proyecto

El desarrollo del proyecto CirKit no requirió inversión monetaria directa gracias a la utilización de herramientas de código abierto y equipos personales. Los únicos costos asociados fueron el tiempo de desarrollo del equipo y el uso de recursos computacionales existentes, making this a zero-budget educational project.

Desglose de Recursos Invertidos:

Horas de Desarrollo: 180 horas totales entre 5 integrantes

Recursos Computacionales: Equipos personales sin costo adicional

Licencias Software: Todas las herramientas fueron de uso libre

Infraestructura: GitHub proporcionó hosting gratuito del código

| Recurso | Especificación | Costo | Proveedor |
|---|---|---|---|
| Laptop Desarrollo 1 | AMD Ryzen 7, 16GB RAM | $0 (Equipo existente) | ASUS |
| Laptop Desarrollo 2 | AMD Athlon, 8GB RAM | $0 (Equipo existente) | |
| Laptop Desarrollo 3 | AMD Ryzen 5, 8GB RAM | $0 (Equipo existente) | ASUS |
| Laptop Desarrollo 4 | Intel Core  i13, 8GB RAM | $0 (Equipo existente) | Lenovo |
| Laptop Desarrollo 5 | Intel Core i31 4GB RAM | $0 (Equipo existente) | Sony |
| Software IDE | Visual Studio Code | $0 (Open Source) | Microsoft |
| Control Versiones | GitHub Repository | $0 (Plan Free) | GitHub |
| Frameworks | Kivy, Tkinter, Python | $0 (Open Source) | Python Software Foundation+ |

*Tabla 11: Recursos del proyecto*

# 10.5 Estándares y Control de Calidad

## 10.5.1 Estándares de Desarrollo Aplicados

El proyecto CirKit ha implementado exhaustivamente estándares internacionales IEEE/ISO/IEC a lo largo de todo su desarrollo, garantizando calidad, confiabilidad y mantenibilidad del software educativo. Para la especificación de requisitos se aplicó el estándar IEEE 830-1998 (Software Requirements Specification), documentando claramente componentes, interfaces y comportamiento de circuitos. El diseño arquitectónico siguió IEEE 1016-2017 (Software Design Description) mediante módulos de simulación bien estructurados que facilitan la colaboración entre desarrolladores. Los procesos de ciclo de vida se estandarizaron con IEEE 12207-2017 (Software Life Cycle Processes), estableciendo fases definidas para diseño, pruebas y actualizaciones del sistema.

En el ámbito de calidad se implementó ISO/IEC 25010:2011 para características de producto software como funcionalidad, rendimiento, usabilidad y fiabilidad. El mantenimiento del código sigue IEEE 14764-2006 (Software Maintenance) con prácticas establecidas para actualizaciones, corrección de errores y mejoras funcionales. Las revisiones técnicas aplican IEEE 1028-2008 (Software Reviews and Audits) mediante evaluaciones periódicas de código que previenen fallos en la simulación de circuitos. La evaluación de procesos utiliza ISO/IEC

15504 (SPICE) para medir madurez del desarrollo e identificar áreas de mejora continua en el ciclo de vida del proyecto.

Las pruebas de software implementan ISO/IEC 29119:2013 mediante planes estructurados que garantizan resultados confiables y reproducibles en todas las simulaciones. La gestión de calidad sigue ISO 9001:2015 para mantener control continuo en actualizaciones y nuevas versiones del simulador. Complementariamente, el código fuente aplica IEEE 754-2019 para operaciones de punto flotante en módulos como checks.py con tolerancias estandarizadas (rtol=1e-6, atol=1e-9), mientras que el núcleo de simulación en tableau.py implementa IEEE 1801-2018 para modelado de sistemas mediante análisis nodal certificado por estándares industriales de ingeniería eléctrica.

La estructura de datos en results.py utiliza ISO/IEC 25010 mediante @dataclass para inmutabilidad y claridad en resultados, mientras que solver.py aplica IEEE 754-2008 para operaciones matriciales con NumPy que aseguran estabilidad numérica. La validación de entradas en todos los módulos sigue IEEE 730-2014 para aseguramiento de calidad, completando un ecosistema de estándares que posiciona a CirKit como software educativo de clase profesional con fundamentos técnicos sólidos y procesos de desarrollo certificados internacionalmente.

### 10.5.2 Sistema de Monitoreo con KPIs

El equipo implementó un sistema integral de monitoreo basado en Indicadores Clave de Rendimiento (KPIs) para evaluar y garantizar la calidad del proyecto durante todas las fases de desarrollo. Este sistema permitió medir objetivamente el avance técnico, la estabilidad del software y la eficiencia en la resolución de problemas. Los KPIs fueron diseñados específicamente para alinearse con los objetivos del proyecto educativo y las capacidades del equipo de desarrollo, proporcionando métricas accionables para la mejora continua.

Los indicadores principales incluyeron mediciones de fiabilidad de cálculos, eficiencia en desarrollo de interfaces, estabilidad de componentes y capacidad de respuesta ante incidentes. Cada KPI contó con responsables definidos, unidades de medida claras y objetivos cuantificables establecidos en cronogramas específicos. La recolección sistemática de datos permitió identificar áreas de mejora y tomar decisiones informadas para optimizar recursos y esfuerzos del equipo durante el ciclo de desarrollo del simulador.

### 10.5.3 Procesos de Verificación y Validación

El proyecto implementó procesos rigurosos de verificación y validación para asegurar que el software cumpliera con los requisitos establecidos y las expectativas de los usuarios finales. La verificación técnica incluyó revisiones de código, pruebas unitarias y validación de algoritmos contra soluciones analíticas conocidas de circuitos eléctricos. La validación funcional abarcó pruebas de usabilidad con usuarios potenciales, evaluando la intuitividad de las interfaces y la claridad de los resultados proporcionados por el sistema.

Se establecieron protocolos específicos para la gestión de incidentes críticos, con tiempos de respuesta definidos y procedimientos de escalamiento para problemas complejos. Las pruebas de estrés evaluaron el comportamiento del sistema bajo condiciones extremas de uso, mientras que las pruebas de regresión aseguraron que nuevas funcionalidades no afectarán componentes existentes. Este enfoque sistemático garantiza la entrega de un producto estable, confiable y alineado con los objetivos educativos del proyecto CirKit.

# 11. Conclusiones y Trabajo Futuro

## 11.1 Conclusiones

El proyecto CirKit ha cumplido exitosamente sus objetivos de desarrollar un simulador de circuitos eléctricos educativo, funcional y accesible. A través de este proyecto, se han alcanzado los siguientes logros:

**Logros Técnicos:**

1. Implementación correcta de algoritmos de cálculo para la Ley de Ohm, resistencias en serie y paralelo, y potencia eléctrica
2. Diseño de una arquitectura modular que facilita el mantenimiento y la extensión del sistema
3. Desarrollo exitoso de tres interfaces de usuario distintas (consola, Tkinter, Kivy), demostrando la flexibilidad de la arquitectura
4. Validación completa de cálculos con 100% de casos de prueba exitosos

**Logros en Gestión de Proyectos:**

1. Aplicación efectiva de metodologías ágiles adaptadas al contexto académico
2. Uso exitoso de Git y GitHub para control de versiones y colaboración
3. Coordinación efectiva de un equipo de 5 integrantes con roles definidos
4. Cumplimiento del cronograma establecido con entrega puntual

## 11.2 Lecciones Aprendidas

Durante el desarrollo del proyecto, el equipo adquirió valiosas lecciones:

1. **Importancia de la planificación:** Una arquitectura bien diseñada desde el inicio facilita enormemente el desarrollo posterior
2. **Valor de la modularidad:** La separación entre lógica y presentación permitió desarrollar múltiples interfaces sin duplicar código
3. **Necesidad de documentación:** La documentación continua es más efectiva que documentar al final del proyecto
4. **Control de versiones:** Git resultó esencial para trabajo colaborativo sin pérdida de código
5. **Pruebas tempranas:** Detectar errores temprano reduce significativamente el tiempo de corrección

## 11.3 Trabajo Futuro

El proyecto CirKit tiene potencial para expansión en múltiples direcciones:

### 11.3.1 Funcionalidades Adicionales Propuestas

**Análisis de Circuitos AC:**

1. Soporte para capacitores e inductores
2. Cálculo de impedancia compleja
3. Análisis en dominio de frecuencia
4. Diagramas de Bode

**Análisis de Circuitos Complejos:**

1. Análisis de mallas (método de corrientes de malla)
2. Análisis nodal (método de tensiones de nodo)
3. Teoremas de Thévenin y Norton
4. Principio de superposición

**Visualización:**

1. Editor gráfico de esquemáticos de circuitos
2. Visualización de formas de onda
3. Gráficas de voltaje vs corriente
4. Representación visual de divisores de voltaje

**Exportación de Datos:**

1. Generación de reportes en PDF
2. Exportación a hojas de cálculo (Excel, CSV)
3. Guardado de circuitos en formato JSON
4. Historial de cálculos

### 11.3.2 Mejoras Técnicas Propuestas

**Optimización de Rendimiento:**

1. Cálculos paralelos para circuitos grandes
2. Caché de resultados frecuentes
3. Optimización de algoritmos

**Mejoras de Interfaz:**

1. Modo oscuro para reducir fatiga visual
2. Temas personalizables
3. Soporte para múltiples idiomas (internacionalización)
4. Tutoriales interactivos integrados

**Despliegue:**

1. Aplicación móvil nativa para Android
2. Aplicación móvil nativa para iOS
3. Versión web usando frameworks como Flask o Django
4. Instalador para Windows (.exe)

### 11.3.3 Extensiones Educativas

**Modo de Aprendizaje:**

1. Ejercicios guiados paso a paso
2. Verificación automática de soluciones de ejercicios
3. Sistema de puntuación y progreso
4. Explicaciones teóricas integradas

**Banco de Problemas:**

1. Colección de problemas de ejemplo
2. Diferentes niveles de dificultad
3. Soluciones detalladas
4. Problemas generados aleatoriamente

## 11.4 Reflexión Final

CirKit representa no solo una herramienta de software funcional, sino también una experiencia de aprendizaje integral en ingeniería de software, gestión de proyectos y trabajo colaborativo. El proyecto demuestra que es posible crear software educativo de calidad utilizando tecnologías de código abierto y metodologías ágiles, incluso en un contexto académico con tiempo limitado.

La arquitectura modular y la documentación completa aseguran que el proyecto pueda continuar evolucionando, ya sea por el equipo original o por futuros colaboradores. CirKit tiene el potencial de convertirse en una herramienta de referencia para la educación en circuitos eléctricos básicos.

# 12. Referencias


Alexander, C. K., & Sadiku, M. N. O. (2020). *Fundamentals of Electric Circuits* (7th ed.). McGraw-Hill Education.

Boylestad, R. L. (2022). *Introductory Circuit Analysis* (14th ed.). Pearson.

Dorf, R. C., & Svoboda, J. A. (2021). *Introduction to Electric Circuits* (10th ed.). John Wiley & Sons.

Hayt, W. H., Kemmerly, J. E., & Durbin, S. M. (2018). *Engineering Circuit Analysis* (9th ed.). McGraw-Hill Education.

Nilsson, J. W., & Riedel, S. A. (2019). *Electric Circuits* (11th ed.). Pearson Education.

Ohm, G. S. (1827). *Die galvanische Kette, mathematisch bearbeitet*. T. H. Riemann.

Van Rossum, G., & Drake, F. L. (2023). *Python 3.13 Documentation*. Python Software Foundation. https://docs.python.org/3.13/

# 13. Apéndices

## Apéndice A: Glosario de Términos

**Amperio (A):** Unidad de medida de corriente eléctrica en el Sistema Internacional.

**API (Application Programming Interface):** Conjunto de definiciones y protocolos para construir e integrar software.

**Backend:** Capa de lógica de negocio de una aplicación, no visible para el usuario final.

**Branch (Rama):** Línea independiente de desarrollo en un sistema de control de versiones.

**Commit:** Confirmación de cambios en un repositorio de control de versiones.

**Framework:** Marco de trabajo que proporciona una estructura y herramientas para desarrollo de software.

**Frontend:** Capa de presentación de una aplicación, visible e interactiva para el usuario.

**Git:** Sistema de control de versiones distribuido para rastrear cambios en código fuente.

**GUI (Graphical User Interface):** Interfaz gráfica de usuario que permite interacción visual.

**Impedancia:** Oposición al flujo de corriente alterna en un circuito, medida en ohmios.

**Kivy:** Framework de Python para desarrollo de aplicaciones multiplataforma con GUI modernas.

**Ohmio (Ω):** Unidad de medida de resistencia eléctrica.

**Pull Request:** Solicitud para fusionar cambios de una rama a otra en Git.

**Python:** Lenguaje de programación interpretado de alto nivel.

**Resistencia:** Oposición al flujo de corriente eléctrica, medida en ohmios.

**Tkinter:** Biblioteca estándar de Python para crear interfaces gráficas de usuario.

**Volt (V):** Unidad de medida de diferencia de potencial eléctrico.

# Apéndice B: Tabla de Conversión de Unidades

| Magnitud | Unidad | Equivalencias |
|---|---|---|
| Resistencia | 1 MΩ | 1,000,000 Ω |
| | 1 kΩ | 1,000 Ω |
| | 1 mΩ | 0.001 Ω |
| Corriente | 1 mA | 0.001 A |
| | 1 µA | 0.000001 A |
| | 1 kA | 1,000 A |
| Voltaje | 1 kV | 1,000 V |
| | 1 mV | 0.001 V |
| | 1 µV | 0.000001 V |
| Potencia | 1 kW | 1,000 W |
| | 1 mW | 0.001 W |
| | 1 MW | 1,000,000 W |

*Tabla 12: Conversiones de unidades eléctricas comunes*

# Apéndice C: Valores Estándar de Resistencias

**Serie E12 (tolerancia 10%):**

10, 12, 15, 18, 22, 27, 33, 39, 47, 56, 68, 82

**Serie E24 (tolerancia 5%):**

10, 11, 12, 13, 15, 16, 18, 20, 22, 24, 27, 30, 33, 36, 39, 43, 47, 51, 56, 62, 68, 75, 82, 91

Estos valores se repiten en múltiplos de 10 (ej: 100Ω, 1kΩ, 10kΩ).

# Apéndice D: Contacto y Soporte

**Repositorio GitHub:**

https://github.com/LaukazBiron/CirKit

**Reporte de Errores:**

https://github.com/LaukazBiron/CirKit/issues

**Curso:**

Administración de Proyectos

**Equipo:**

C09676 Studios

# Glosario de estándares ISO

| CÓDIGO | NOMBRE DEL ESTÁNDAR | APLICACIÓN EN CIRKIT | PARTE DEL SISTEMA DONDE SE APLICA |
|---|---|---|---|
| IEEE 830-1998 | Software Requirements Specification (SRS) | Documenta requisitos de componentes, interfaz y comportamiento de circuitos | Especificación inicial y documentación de requisitos |
| IEEE 1016-2017 | Software Design Description (SDD) | Define arquitectura modular del código y relaciones entre módulos | Diseño de arquitectura y documentación técnica |
| IEEE 12207-2017 | Software Life Cycle Processes | Establece fases para diseñar, probar y actualizar la aplicación | Gestión completa del ciclo de vida del proyecto |
| IEEE 14764-2006 | Software Maintenance | Guía actualizaciones de código, corrección de errores y mejoras | Mantenimiento y actualizaciones del sistema |
| IEEE 1028-2008 | Software Reviews and Audits | Realiza revisiones de código y auditorías técnicas para calidad | Procesos de revisión y control de calidad |
| ISO/IEC 25010:2011 | Calidad del Producto de Software | Define características de funcionalidad, rendimiento y usabilidad | Criterios de calidad en todos los módulos |
| ISO/IEC 12207:2017 | Procesos del Ciclo de Vida del Software | Documenta y controla etapas de desarrollo y mantenimiento | Gestión de procesos de desarrollo |
| ISO/IEC 15504 (SPICE) | Evaluación de Procesos de Software | Mide la madurez del proceso de desarrollo e identificar mejoras | Evaluación y mejora de procesos |
| ISO/IEC 29119:2013 | Pruebas de Software | Planifica y documenta pruebas de simulación para resultados confiables | Suite de pruebas y validación |
| ISO 9001:2015 | Gestión de Calidad | Mantiene control de calidad continuo en actualizaciones | Sistema de gestión de calidad |
| IEEE 754-2019 | Floating-Point Arithmetic | Garantiza precisión en operaciones numéricas de simulaciones | Módulos checks.py, tableau.py, solver.py |

| | | | |
|---|---|---|---|
| IEEE 1801-2018 | Design and Verification of Low-Power Systems | Modela sistemas de potencia mediante análisis nodal | Núcleo de simulación en tableau.py |
| IEEE 730-2014 | Software Quality Assurance | Valida entradas y asegura calidad del código | Validación en todos los módulos |
| ISO/IEC 11179-2015 | Specification of Metadata | Estructura metadatos de simulación y componentes | Metadatos en tableau.py y results.py |
| ISO/IEC/IEEE 42010-2011 | Architecture Description | Define arquitectura del sistema y sus componentes | Arquitectura general del sistema |
| IEEE 1016-2009 | Software Design Descriptions | Documenta diseño detallado de módulos y componentes | Documentación de diseño técnico |
| ISO/IEC 10967 | Language Independent Arithmetic | Establece tolerancias numéricas para cálculos confiables | Módulo checks.py con rtol/atol |
| ISO/IEC 8652-2012 | Ada Language Reference | Aplica anotaciones de tipo para mejor documentación | Anotaciones de tipo en Python |
| IEEE 1076-2008 | VHDL Language Reference Manual | Modela componentes electrónicos y sus conexiones | Componentes en domain/ |
| IEEE 1666-2011 | SystemC Language Reference Manual | Estructura simulación de sistemas electrónicos | Sistema de simulación general |

*Tabla 13: Conversiones de unidades eléctricas comunes*

# Glosario KPI

| Término | Definición | Contexto en CirKit |
|---|---|---|
| KPI (Indicador Clave de Rendimiento) | Métrica cuantificable que mide el desempeño de procesos críticos para el éxito del proyecto | Indicadores establecidos para evaluar calidad, tiempo y funcionalidad del simulador |
| Fiabilidad de Cálculos | Porcentaje que representa la precisión de los resultados generados por el sistema | Medición de exactitud en simulaciones de circuitos comparado con valores teóricos esperados |
| Tiempo de Implementación | Período requerido para completar el desarrollo de componentes específicos | Horas dedicadas a crear interfaces de usuario y módulos de funcionalidad |
| Estabilidad de Componentes | Capacidad del software para funcionar sin fallos durante operaciones continuas | Evaluación de rendimiento en pruebas prolongadas de simulación de circuitos |
| Velocidad de Resolución | Tiempo transcurrido entre la detección de un problema y su solución completa | Rapidez para corregir errores críticos que afectan la funcionalidad del simulador |
| Términos de Medición y Análisis | | |
| Término | Definición | Contexto en CirKit |
| Porcentaje de Fiabilidad | Valor porcentual que indica el nivel de confianza en los resultados del sistema | KPI principal que mide la precisión de cálculos de circuitos eléctricos |
| Horas Hombre | Unidad de medida que representa el trabajo realizado por una persona en una hora | Métrica para cuantificar esfuerzo en desarrollo de interfaces y funcionalidades |
| Tasa de Estabilidad | Relación entre operaciones exitosas y total de operaciones realizadas | Medición de fallos en simulaciones consecutivas del sistema |
| Desviación Acumulada | Diferencia total entre lo planificado y lo realmente ejecutado en hitos | Control de cumplimiento de fechas críticas en el cronograma de desarrollo |

| | | |
|---|---|---|
| Tiempo de Respuesta | Período entre la solicitud de una acción y su ejecución completa | Velocidad para resolver incidentes críticos reportados por usuarios |
| Términos de Gestión de Calidad | | |
| Término | Definición | Contexto en CirKit |
| Objetivo General KPI | Meta principal que define el propósito fundamental del sistema | Garantizar funcionalidad básica y confiabilidad del simulador de circuitos |
| Objetivo Específico KPI | Meta particular enfocada en aspectos concretos del desarrollo | Optimización de tiempos de desarrollo y estabilidad de componentes |
| Unidad de Medida | Estándar cuantitativo utilizado para expresar el valor de un KPI | Porcentajes, horas, días o tasas que permiten evaluación objetiva |
| Solicitante KPI | Rol responsable de definir y requerir el seguimiento del indicador | Integrante que identifica la necesidad de medir aspecto específico |
| Ejecutor KPI | Rol encargado de implementar y dar seguimiento al indicador | Integrante responsable de recopilar datos y generar reportes |
| Términos Técnicos Especializados | | |
| Término | Definición | Contexto en CirKit |
| Circuito Ideal | Modelo teórico que no considera imperfecciones de componentes | Casos de prueba para validar precisión de algoritmos de simulación |
| Circuito Real | Modelo que incluye tolerancias y características no ideales | Escenarios avanzados para evaluar robustez del simulador |
| Tolerancia Numérica | Margen de error aceptable en cálculos computacionales | Parámetros rtol/atol en verificaciones de leyes de Kirchhoff |
| Incidente Crítico | Problema que afecta significativamente la funcionalidad del sistema | Errores que impiden la simulación correcta de circuitos básicos |

| | | |
|---|---|---|
| Hito de Proyecto | Evento significativo que marca avance importante en desarrollo | Entregas de versiones, módulos completados y pruebas exitosas |

# Índice de Contenidos

**1. Introducción.....2**
1.1 Contexto del Proyecto.....2
1.2 Objetivos del Sistema.....2

1.2.1 Objetivo General..... 2
1.2.2 Objetivos Específicos..... 2
1.2.2.1 KPI de Objetivo General..... 3
1.2.2.2 KPI Objetivo Específico 1..... 3
1.2.2.3 KPI Objetivo Específico 2..... 4
1.2.2.4 KPI Objetivo Específico 3..... 4
1.2.2.5 KPI de Desempeño..... 5
1.3 Alcance y Limitaciones..... 5
1.3.1 Alcance..... 5
1.3.2 Limitaciones..... 6
1.4 Audiencia Objetivo..... 6
**2. Fundamentos Teóricos..... 7**
2.1 Ley de Ohm..... 7
2.1.1 Aplicación en CirKit..... 7
2.2 Resistencias en Serie..... 8
2.2.1 Propiedades de las Resistencias en Serie..... 8
2.3 Resistencias en Paralelo..... 8
2.3.1 Propiedades de las Resistencias en Paralelo..... 9
**3. Arquitectura del Sistema..... 10**
3.1 Diseño Arquitectónico..... 10
3.2 Modelo de Capas..... 10
Tabla 1: Arquitectura de capas del sistema CirKit..... 11
3.3 Diagrama de Componentes..... 11
3.4 Patrones de Diseño Aplicados..... 11
3.4.1 Patrón MVC (Modelo-Vista-Controlador)..... 12
3.4.2 Patrón de Módulos..... 12
3.4.3 Principio DRY (Don't Repeat Yourself)..... 12
**4. Especificación Técnica..... 13**
4.1 Requisitos del Sistema..... 13
4.1.1 Requisitos de Hardware..... 13
Tabla 2: Requisitos mínimos de hardware..... 13
4.1.2 Requisitos de Software..... 13
Tabla 3: Requisitos de software..... 14
4.2 Tecnologías Utilizadas..... 14
4.2.1 Lenguaje de Programación..... 14
4.2.2 Bibliotecas y Frameworks..... 14
4.3 Estructura del Proyecto..... 15
Imagen 1. Diagrama de paquetes sobre la estructura general del proyecto... 16
4.3.1 Descripción de la Arquitectura de Software..... 16

4.3.2 Estructura en el repositorio..... 19
4.4 Dependencias..... 19
4.4.1 Archivo requirements.txt..... 19
4.4.2 Instalación de Dependencias..... 20
**5. Módulos y Componentes..... 21**
Imagen 2. Diagrama UML sobre el motor de cálculo (netlist) y validación del proyecto..... 21
5.1 Módulo de Lógica de Circuitos (logica_circuitos.py)..... 21
5.1.1 Funciones Principales..... 21
5.1.2 Funciones de Validación..... 23
5.2 Módulo de Interfaz Tkinter (interfaz_tkinter.py)..... 24
5.2.1 Componentes de la GUI..... 24
5.2.2 Eventos y Callbacks..... 26
5.3 Módulo de Interfaz Kivy (interfaz_kivy.py)..... 27
5.3.1 Arquitectura Kivy..... 28
5.3.2 Widgets Utilizados..... 29
5.3.3 Ejemplo de Código KV..... 29
**6. Metodología de Desarrollo..... 30**
Imagen 3. Diagrama de casos de uso para el usuario del proyecto..... 30
6.1 Gestión de Proyecto..... 30
6.1.1 Metodología Aplicada..... 30
6.1.2 Herramientas de Gestión..... 31
6.2 Control de Versiones..... 31
6.2.1 Sistema de Control de Versiones..... 31
6.2.2 Estrategia de Branching..... 31
6.3 Flujo de Trabajo con Git..... 32
6.3.1 Comandos Básicos Utilizados..... 32
6.3.2 Convenciones de Commits..... 33
6.4 Roles y Responsabilidades..... 34
6.4.1 Equipo de Desarrollo..... 34
Tabla 4: Equipo de desarrollo y responsabilidades..... 34
6.4.2 Matriz de Responsabilidades RACI..... 35
Tabla 5: Matriz RACI (R=Responsable, A=Aprobador, C=Consultado, I=Informado). 35
**7. Guía de Instalación y Configuración..... 36**
7.1 Requisitos Previos..... 36
7.1.1 Verificación de Python..... 36
7.1.2 Verificación de pip..... 36
7.1.3 Verificación de Git (Opcional)..... 37
7.2 Instalación Paso a Paso..... 37

7.2.1 Método 1: Clonar desde GitHub (Recomendado)..... 37
7.2.2 Método 2: Descarga Manual..... 37
7.3 Configuración del Entorno..... 38
7.3.1 Entorno Virtual (Recomendado)..... 38
7.4 Verificación de la Instalación..... 38
7.4.1 Prueba de la Interfaz de Consola..... 39
7.4.2 Prueba de la Interfaz Tkinter..... 39
7.4.3 Prueba de la Interfaz Kivy..... 39
7.4.4 Solución de Problemas Comunes..... 39
**8. Pruebas y Validación..... 40**
8.1 Estrategia de Pruebas..... 40
8.1.1 Niveles de Prueba..... 40
8.1.2 Herramientas de Prueba..... 40
8.2 Casos de Prueba..... 40
8.2.1 Pruebas de la Ley de Ohm..... 40
Tabla 6: Casos de prueba para la Ley de Ohm..... 41
8.2.2 Pruebas de Resistencias en Serie..... 41
Tabla 7: Casos de prueba para resistencias en serie..... 41
8.2.3 Pruebas de Resistencias en Paralelo..... 41
Tabla 8: Casos de prueba para resistencias en paralelo..... 41
8.2.4 Pruebas de Potencia Eléctrica..... 41
Tabla 9: Casos de prueba para cálculo de potencia..... 42
8.3 Resultados de Validación..... 42
8.3.1 Cobertura de Pruebas..... 42
8.3.2 Validación con Valores Conocidos..... 42
8.3.3 Pruebas de Usabilidad..... 42
**9. Mantenimiento y Extensión..... 43**
9.1 Agregar Nuevas Funcionalidades..... 43
9.1.1 Procedimiento para Agregar una Nueva Función de Cálculo..... 43
9.1.2 Ejemplo: Agregar Soporte para Inductores..... 44
9.2 Modificar Componentes Existentes..... 44
9.2.1 Mejora de Validación de Entrada..... 44
9.2.2 Mejora de Formato de Salida..... 45
9.3 Guía de Contribución..... 46
9.3.1 Proceso de Contribución..... 46
9.3.2 Estándares de Código..... 46
**10. Gestión de Proyecto..... 48**
10.1 Cronograma de Desarrollo..... 48
Tabla 10: Cronograma realista de desarrollo del proyecto..... 48

10.2 Fases del Proyecto..... 48
10.2.1 Fase 1: Diseño de la Interfaz Gráfica..... 48
10.2.2 Fase 2: Implementación de 3 Componentes Básicos..... 49
10.2.3 Fase 3: Supervisar Pruebas y Correcciones..... 49
10.3 Entregables..... 50
10.3.1 Entregables de Software..... 50
10.3.2 Entregables de Documentación..... 50
10.3.3 Entregables de Gestión..... 50
10.4 Recursos y Equipos de Desarrollo..... 50
10.4.1 Infraestructura de Hardware..... 50
10.4.2 Software y Herramientas de Desarrollo..... 51
10.4.3 Presupuesto y Costos del Proyecto..... 51
Tabla 11: Recursos del proyecto..... 52
10.5 Estándares y Control de Calidad..... 52
10.5.1 Estándares de Desarrollo Aplicados..... 52
10.5.2 Sistema de Monitoreo con KPIs..... 53
10.5.3 Procesos de Verificación y Validación..... 54
**11. Conclusiones y Trabajo Futuro..... 55**
11.1 Conclusiones..... 55
11.2 Lecciones Aprendidas..... 55
11.3 Trabajo Futuro..... 56
11.3.1 Funcionalidades Adicionales Propuestas..... 56
11.3.2 Mejoras Técnicas Propuestas..... 56
11.3.3 Extensiones Educativas..... 57
11.4 Reflexión Final..... 57
**12. Referencias..... 58**
**13. Apéndices..... 59**
Apéndice A: Glosario de Términos..... 59
Apéndice B: Tabla de Conversión de Unidades..... 60
Tabla 12: Conversiones de unidades eléctricas comunes..... 60
Apéndice C: Valores Estándar de Resistencias..... 60
Apéndice D: Contacto y Soporte..... 60
**Glosario de estándares ISO..... 62**
Tabla 13: Conversiones de unidades eléctricas comunes..... 63
**Glosario KPI..... 64**