



INSTITUTO POLITÉCNICO NACIONAL

---

---

UNIDAD PROFESIONAL  
INTERDISCIPLINARIA DE INGENIERÍA Y  
CIENCIAS SOCIALES Y ADMINISTRATIVAS

**6NM60 Ingeniería de Pruebas**

**Desarrollo de un sistema: Calculadora digital  
intuitiva y multifuncional.**

Alumnos:

García Méndez Juan Carlos

Conde Basilio Leonardo

Felipe

Enrique

Docente:

Cruz Martínez Ramón

Fecha: 04 de marzo del 2025



## Contenido

Misión.....	3
Visión.....	3
Marco Teórico .....	4
Planteamiento del Problema: .....	5
Objetivo:.....	5
Hipótesis:.....	6
Justificación .....	6
Marco Metodológico:.....	7
Desarrollo y Solución:.....	7
Forma de Trabajo: .....	8
Levantamiento de Requisitos:.....	9
Desarrollo de interfaces .....	10
Nuevos Requerimientos.....	16
Requisitos Funcionales (RF): .....	16
Requisitos No Funcionales (RNF): .....	16
Actualizaciones en Otras Secciones .....	17
12.5 Riesgos y Mitigación (Añadidos) .....	18
Integración Técnica .....	18

## Misión

Desarrollar una calculadora digital intuitiva y multifuncional que integre operaciones matemáticas básicas, gráficos simples en dos ejes y gestión de historial, garantizando precisión (hasta dos decimales), manejo de errores (como división entre cero) y una experiencia de usuario fluida. Nuestra misión es proporcionar una herramienta educativa y práctica que satisfaga las necesidades de estudiantes, profesionales y usuarios ocasionales, priorizando la claridad, la accesibilidad y la robustez técnica.

## Visión

Posicionarnos como una herramienta de referencia en el ámbito educativo y profesional, destacando por nuestra capacidad para combinar funcionalidades avanzadas (como gráficos e historial) con una interfaz familiar (similar al teclado de un teléfono). Aspiramos a evolucionar continuamente, incorporando tecnologías emergentes y ampliando capacidades, siempre bajo los principios de calidad, innovación y accesibilidad universal.

## Marco Teórico

### Antecedentes:

Las calculadoras digitales han sido fundamentales desde los años 1960, evolucionando de dispositivos básicos a aplicaciones multifuncionales. Sin embargo, muchas herramientas actuales carecen de integración de funciones gráficas o historial accesible, limitando su utilidad en contextos educativos.

### Funcionalidades y Base Técnica:

- **Operaciones básicas:** Implementación de algoritmos matemáticos estándar (suma, resta, multiplicación, división) con manejo de números negativos y redondeo a dos decimales.
- **División entre cero:** Incorporación de excepciones controladas para evitar errores críticos, mostrando mensajes claros (ej: "Error: División entre cero").
- **Gráficos 2D:** Uso de bibliotecas gráficas para representar funciones ingresadas por el usuario.
- **Historial:** Almacenamiento temporal en memoria o archivos locales para visualización sin edición.
- **Interfaz de usuario:** Diseño basado en el teclado numérico telefónico (3x3 + 0) para reducir la curva de aprendizaje.

### Relevancia:

La integración de gráficos y operaciones básicas en una sola herramienta responde a la necesidad de visualización inmediata de resultados, crucial en educación STEM. Además, el historial y el borrado selectivo mejoran la eficiencia en cálculos secuenciales.

## Planteamiento del Problema:

Las calculadoras tradicionales y aplicaciones móviles suelen fragmentar funcionalidades:

- No integran gráficos con operaciones básicas.
- Manejan errores como división entre cero de forma críptica.
- Carecen de historial visible o permiten editar resultados previos.
- Interfaces complejas que dificultan la adopción por nuevos usuarios.

Esta fragmentación genera frustración en usuarios que requieren una herramienta unificada para resolver problemas matemáticos cotidianos o académicos, especialmente en entornos educativos donde la retroalimentación visual (gráficos) y la trazabilidad (historial) son esenciales.

## Objetivo:

Desarrollar una calculadora multifuncional que:

- Ejecute operaciones básicas y maneje números negativos/decimales.
- Genere gráficos 2D a partir de ecuaciones simples.
- Registre y muestre un historial de operaciones no editable.
- Incorpore un sistema de borrado selectivo (total o por dígito).
- Garantice usabilidad mediante una interfaz familiar y respuestas inmediatas.

## Hipótesis:

Si se integran operaciones matemáticas básicas, gráficos 2D y un historial visible en una interfaz intuitiva (similar a un teléfono), entonces los usuarios podrán resolver problemas matemáticos cotidianos y académicos con mayor eficiencia, reduciendo el tiempo dedicado a cambiar entre herramientas y mejorando la comprensión mediante visualización gráfica.

## Justificación

### Educativa:

- Los gráficos ayudan a estudiantes a relacionar operaciones abstractas con representaciones visuales.
- El historial permite revisar pasos previos, útil en corrección de errores.

### Técnica:

- El manejo de decimales y negativos asegura precisión en cálculos financieros o científicos.
- La doble opción de borrado optimiza la interacción (ej: corrección rápida de un dígito erróneo).

### Social:

- Una herramienta gratuita y accesible reduce la brecha tecnológica en comunidades con recursos limitados.

### Calidad:

- La documentación exhaustiva (requerimientos, casos de prueba) garantiza mantenibilidad y escalabilidad, clave en ingeniería de software.

## Marco Metodológico:

La metodología empleada combina enfoques **ágiles** (para desarrollo iterativo) y **basados en pruebas** (TDD - *Test-Driven Development*), adaptados a un proyecto académico.

- **Fases:**

1. **Análisis de requerimientos:** Validación y priorización de funcionalidades (ej: gráficos vs. historial).
2. **Diseño modular:** Separación en componentes: interfaz, motor de cálculos, módulo gráfico y gestor de historial.
3. **Desarrollo incremental:** Construcción por iteraciones semanales, integrando una funcionalidad principal por ciclo (ej: operaciones básicas en la primera iteración).
4. **Pruebas continuas:** Cada módulo se valida con casos de prueba unitarios y de integración (ej: probar división entre cero antes de implementar gráficos).
5. **Retroalimentación:** Simulación de escenarios con usuarios reales (estudiantes y profesores) para ajustar la interfaz.

- **Herramientas:**

- *Java* (lenguaje principal).
- *Git* (control de versiones).

## Desarrollo y Solución:

Estrategia Técnica:

1. Motor de Cálculos:

- Operaciones básicas: Uso de funciones matemáticas nativas de Python, con encapsulamiento en una clase Calculadora para manejar decimales (`round(result, 2)`) y negativos.
- Manejo de errores: Implementación de excepciones personalizadas (ej: `raise ZeroDivisionError("División entre cero no permitida")`).

2. Gráficos 2D:

- Integración de Matplotlib para generar gráficos a partir de ecuaciones ingresadas (ej:  $y = 2x + 3$ ).
- Validación de entradas mediante expresiones regulares para evitar inyección de código.

3. Historial de Operaciones:

- Almacenamiento en una lista de diccionarios con estructura `{operacion: "3+5", resultado: "8", fecha: "2024-05-20"}`.
- Visualización en una ventana secundaria con scroll, bloqueando la edición (solo lectura).

4. Interfaz de Usuario:

- Diseño de teclado telefónico (3x4) con Tkinter, usando Grid Layout para alinear botones numéricos y funcionales.
- Botones de borrado:
- CE (Borrar Todo): Limpia la pantalla y reinicia variables.
- ← (Borrar último dígito): Manipulación de strings (`pantalla_texto = pantalla_texto[:-1]`).

#### 5. Pruebas:

- Unitarias: Verificación de operaciones matemáticas (ej: `assert calcular("2.5 + 3.7") == 6.2`).
- Integración: Flujo completo desde entrada de datos hasta generación de gráficos.
- Usabilidad: Pruebas A/B con dos diseños de interfaz para evaluar eficiencia en la navegación.

## Forma de Trabajo:

### Equipo:

- Roles:
  - *Programador*: Responsable de la arquitectura del código.
  - *Diseñador UI/UX*: Define disposición de botones y paleta de colores.
  - *Tester*: Ejecuta casos de prueba y reporta bugs.
  - *Analista*: Elabora manuales técnicos y de usuario.

### Flujo de Trabajo:

1. Sprints de 1 semana:
  - Lunes: Planificación de tareas.
  - Martes-jueves: Desarrollo y pruebas internas.
  - Viernes: Demostración y ajustes.
2. Comunicación:
  - Reuniones diarias de 15 minutos (Scrum) para sincronizar avances.
  - Uso de *Slack* para consultas rápidas y *Google Drive* para compartir documentos.
3. Control de Calidad:
  - Revisión de código por pares (*peer review*).
  - Checklist de entrega: Funcionalidad probada, documentación actualizada, código comentado.



Levantamiento de Requisitos:

Requisitos Funcionales (RF):

ID	Requisito	Prioridad	Descripción Técnica
RF01	Realizar operaciones básicas	ALTA	Suma, resta, multiplicación y división con soporte para negativos y dos decimales.
RF02	Manejar división entre cero	ALTA	Mostrar mensaje claro sin bloquear la aplicación.
RF03	Generar gráficos 2D	MEDIA	Gráficos de funciones lineales (ej: $y = mx + b$ ) con ejes etiquetados.
RF04	Historial visible	MEDIA	Lista ordenada de las últimas 20 operaciones, sin opción de modificar.
RF05	Borrado selectivo	ALTA	Dos botones: borrar último dígito ( $\leftarrow$ ) y borrar todo (CE).
RF06	Interfaz tipo teclado telefónico	ALTA	Distribución 3x4 con botones numéricos (0-9) y operadores en posiciones estándar.

Requisitos No Funcionales (RNF):

ID	Requisito	Prioridad	Métrica de Cumplimiento
RNF01	Rendimiento	ALTA	Respuesta en <1 segundo para operaciones básicas.
RNF02	Usabilidad	ALTA	90% de usuarios encuestados navegan sin requerir manual.
RNF03	Compatibilidad	MEDIA	Funcionar en Windows 10+, macOS 12+ y navegadores modernos.
RNF04	Seguridad	MEDIA	Validación de entradas para evitar inyección de código.

Casos de Uso Críticos:

- CUC01: Usuario divide  $5/0$  → Sistema muestra "Error: División entre cero".
- CUC02: Usuario ingresa  $2.5 * -3$  → Sistema muestra "-7.50".
- CUC03: Usuario gráfica  $y = x^2$  → Sistema renderiza una parábola en una ventana emergente.

## Desarrollo de interfaces.

### 12. Desarrollo de Interfaces

El diseño de la interfaz es crítico para garantizar una experiencia de usuario intuitiva y eficiente. A continuación, se detallan las herramientas recomendadas, el diseño propuesto y los principios de implementación.

#### 12.1 Herramientas Recomendadas

Para el desarrollo de la interfaz gráfica y sus componentes, se proponen las siguientes herramientas:

<b>Componente</b>	<b>Herramienta</b>	<b>Justificación</b>
<b>Interfaz Principal</b>	<b>Tkinter (Python)</b>	<ul style="list-style-type: none"><li>- Biblioteca nativa de Python, ligera y multiplataforma.</li><li>- Soporta Grid Layout para alinear botones en formato 3x4 (teléfono).</li><li>- Integración sencilla con módulos de cálculos y gráficos.</li></ul>
<b>Gráficos 2D</b>	<b>Matplotlib (Python)</b>	<ul style="list-style-type: none"><li>- Ampliamente usada en proyectos académicos.</li><li>- Permite generar gráficos en ventanas emergentes con ejes etiquetados.</li><li>- Compatible con Tkinter mediante FigureCanvasTkAgg.</li></ul>
<b>Diseño Responsivo</b>	<b>CustomTkinter</b>	<ul style="list-style-type: none"><li>- Librería que moderniza Tkinter con temas actualizados y widgets personalizables.</li><li>- Útil si se prioriza estética profesional.</li></ul>
<b>Prototipado</b>	<b>Figma</b>	<ul style="list-style-type: none"><li>- Herramienta colaborativa para diseñar mockups interactivos.</li><li>- Permite validar la disposición del teclado y flujos de usuario antes de codificar.</li></ul>

### 12.2 Diseño de la Interfaz

#### Estructura Visual:

##### 1. Área de Pantalla:

- **Ubicación:** Superior.
- **Función:** Muestra la entrada actual y los resultados.
- **Características:**
  - Fuente grande (14pt) para legibilidad.
  - Soporte para números negativos (ej: -5.00).

## 2. Teclado Numérico:

- Distribución: 3x4 (como un teléfono móvil):

[7] [8] [9] [ $\div$ ]

[4] [5] [6] [ $\times$ ]

[1] [2] [3] [-]

[0] [.] [=] [ + ]

- **Botones Especiales:**

- **CE (Borrar todo) y  $\leftarrow$  L (Borrar último dígito) en fila adicional superior y (Borrar última operación).**
- **Gráfico e Historial como botones laterales o en menú contextual.**

## 3. Panel de Historial:

- Ubicación: Ventana lateral o emergente.
- Características:
  - Lista de hasta 20 operaciones en formato [fecha]  $5 + 3 = 8.00$ .
  - Scroll vertical para navegar.

### 12.3 Buenas Prácticas de UI/UX

- **Consistencia Visual:**

- Misma paleta de colores para operadores (ej: rojo para  $\div$ ,  $\times$ ).
- Espaciado uniforme entre botones (padding de 5px).

- **Feedback al Usuario:**

- Cambio de color al presionar botones.
- Mensajes de error en rojo y éxito en verde.

- **Accesibilidad:**

- Soporte para navegación con teclado (ej: tecla Enter para =).
- Fuentes de alto contraste (negro sobre gris claro).

## 12.5 Riesgos y Mitigación

Riesgo	Mitigación
<b>Gráficos no se renderizan</b>	<b>Validar ecuaciones antes de procesar para mostrar las imágenes.</b>
<b>Interfaz lenta en equipos viejos</b>	<b>Optimizar uso de recursos (ej: limitar historial a 20 operaciones).</b>
<b>Confusión en disposición de botones</b>	<b>Realizar pruebas de usabilidad con prototipos en Figma antes de codificar.</b>

**Wireframes:** Los wireframes son esquemas básicos y simplificados que representan la estructura y disposición de los elementos de una interfaz, estos esquemas están enfocados en la funcionalidad y la organización permitiendo una visión inicial de la estructura y planificar la jerarquía de información y el flujo de navegación.

**Mockups:** Los mockups son representaciones visuales de alta fidelidad que muestran cómo se verá la interfaz final, incluyendo tipografías, imágenes, íconos y otros elementos visuales, esto tiene el objetivo de identificar problemas visuales antes del desarrollo, así como comunicar la apariencia de la interfaz a los clientes

## Arquitectura y tecnología:

La solución se desarrolló bajo un enfoque modular y en capas, utilizando tecnologías robustas y escalables para garantizar mantenibilidad y eficiencia.

### 14.1 Stack Tecnológico

Componente	Tecnología/Herramienta	Descripción
<b>Lenguaje Principal</b>	<b>Java 17</b>	<ul style="list-style-type: none"><li>- Versión LTS (Long-Term Support) para garantizar estabilidad.</li><li>- Manejo de excepciones, POO y tipos de datos precisos (ej: BigDecimal).</li></ul>
<b>Interfaz Gráfica</b>	<b>Java Swing</b>	<ul style="list-style-type: none"><li>- Biblioteca estándar para GUIs multiplataforma.</li><li>- Uso de JFrame, JPanel, y GridLayout para replicar el teclado telefónico.</li></ul>
<b>IDE</b>	<b>Apache NetBeans 25</b>	<ul style="list-style-type: none"><li>- Entorno integrado con herramientas visuales para diseño de GUIs (arrastrar y soltar).</li></ul>

		- Depurador integrado y soporte para Maven/Gradle.
<b>Gestión de Dependencias</b>	<b>Maven</b>	- Automatización de builds, gestión de librerías y generación de ejecutables.
<b>Pruebas</b>	<b>JUnit 5</b>	- Framework para pruebas unitarias y de integración. - Compatibilidad con NetBeans.
<b>Control de Versiones</b>	<b>Git + GitHub</b>	- Seguimiento de cambios y colaboración en el código.

## 14.2 Diagrama de Arquitectura

Capa de presentación (GUI) → Capa de lógica de negocio (Operaciones) → Capa de datos (historial)

- **Capa de Presentación (GUI):**
  - Ventanas creadas con JFrame y componentes Swing (JButton, JTextField).
  - Diseño del teclado telefónico mediante GridLayout (3 filas x 4 columnas).
- **Capa de Lógica de Negocio:**
  - Clases como Calculadora.java con métodos para operaciones matemáticas (sumar(), dividir()).
  - Manejo de decimales con BigDecimal para precisión y redondeo a dos decimales.
  - Validación de entradas (ej: evitar caracteres no numéricos).
- **Capa de Datos:**
  - Almacenamiento del historial en una lista (LinkedList<String>) con persistencia temporal durante la sesión.
  - Acceso al historial mediante una ventana secundaria (JDialog).

## 14.3 Flujo de Datos

### 1. Entrada del Usuario:

- El usuario interactúa con botones en la GUI.
- Los eventos (ej: clic en botón "7") se gestionan mediante ActionListener.

### 2. Procesamiento:

- La lógica de negocio valida y ejecuta operaciones.
- Los resultados se redondean y formatean antes de enviarse a la GUI.

### 3. Salida:

- Resultados mostrados en un JTextField.
- Historial actualizado en tiempo real.

## 15. Características Generales de la Solución

### 15.1 Funcionalidades Clave

#### 1. Operaciones Matemáticas Básicas:

- Suma, resta, multiplicación y división con soporte para números negativos y decimales (ej:  $-5.25 + 3.75 = -1.50$ ).
- Redondeo automático a **dos decimales** en todos los resultados.

#### 2. Manejo de Errores Robustos:

- Detección de división entre cero con mensaje claro: Error: División no permitida.
- Bloqueo de entradas inválidas (ej: letras o símbolos no numéricos).

#### 3. Historial de Operaciones:

- Visualización de las últimas 20 operaciones en una ventana emergente.
- Formato de registro: [12/05/2024]  $8.50 \div 2.00 = 4.25$ .

#### 4. Borrado Selectivo:

- **Botón "CE":** Borra toda la entrada actual.
- **Botón "←":** Elimina el último dígito ingresado.
- **Botón "L":** Borra la última operación ingresada.

#### 5. Interfaz Intuitiva:

- Distribución de teclado idéntica a un teléfono móvil (3x4).
- Diseño minimalista con colores contrastantes para mejor legibilidad.

### 15.2 Ventajas Técnicas

- **Portabilidad:** Ejecutable en cualquier sistema con JRE instalado (Windows, Linux, macOS).
- **Mantenibilidad:** Código modular (separación GUI/lógica/datos) para facilitar actualizaciones.
- **Rendimiento:** Uso de BigDecimal evita errores de redondeo en operaciones financieras.
- **Seguridad:** Validación de entradas para prevenir errores de formato o inyección de código.

### 15.3 Limitaciones

- **Persistencia de Datos:** El historial no se guarda después de cerrar la aplicación.
- **Complejidad de Operaciones:** No incluye funciones avanzadas (ej: potencias, raíces).

### 16. Integración con NetBeans

- **Diseño Visual:** Uso del editor **Swing GUI Designer** de NetBeans para arrastrar y soltar componentes (botones, campos de texto).
- **Generación de Código Automático:** NetBeans genera el esqueleto de clases (ej: initComponents()).
- **Pruebas Unitarias Integradas:** Configuración directa de JUnit desde el IDE.

## Nuevos Requerimientos

### Requisitos Funcionales (RF):

ID	Requisito	Prioridad	Descripción Técnica
RF07	Botón de porcentaje (%)	MEDIA	Calcular porcentajes sobre el valor ingresado (ej: $100 + 10\% = 110$ ).
RF08	Botón de almacenar en memoria (M+)	ALTA	Guardar el valor actual en una variable de memoria (persistente durante la sesión).
RF09	Sumar a memoria (M+)	MEDIA	Acumular el valor actual al valor almacenado en memoria (ej: $M=5 \rightarrow M+3 \rightarrow M=8$ ).
RF10	Recuperar memoria (MR)	ALTA	Mostrar el valor almacenado en memoria en la pantalla actual.
RF11	Limpiar memoria (MC)	ALTA	Restablecer la memoria a cero.

### Requisitos No Funcionales (RNF):

ID	Requisito	Prioridad	Métrica de Cumplimiento
RNF05	Consistencia de Memoria	ALTA	La memoria debe retener su valor hasta que la aplicación se cierre o se use el botón MC.
RNF02	Usabilidad	ALTA	85% de usuarios encuestados entienden la función de memoria sin consultar manual.



## Actualizaciones en Otras Secciones

### 12.2 Diseño de la Interfaz (Modificado)

- **Nuevos Botones en el Teclado:**
  - Fila adicional inferior para memoria:  
[MC] [MR] [M+] [%]
  - Características:
    - Color diferenciado (ej: azul para memoria).
    - Tooltips al pasar el cursor (ej: "Memoria: 5.00").

### 14.2 Diagrama de Arquitectura (Ajustado)

- **Capa de Lógica de Negocio:**
  - Nueva clase Memoria.java con métodos:
    - almacenar(valor), acumular(valor), recuperar(), limpiar().
  - Integración con Calculadora.java para operaciones que usen memoria.

### 15.1 Funcionalidades Clave (Ampliadas)

#### 6. Gestión de Memoria:

- Almacenamiento temporal de valores durante la sesión.
- Operaciones acumulativas (M+) y recuperación instantánea (MR).
- Feedback visual del estado de la memoria (ej: indicador LED en la pantalla).

### 15.3 Limitaciones (Actualizadas)

- La memoria no es persistente entre sesiones (se pierde al cerrar la aplicación).

### Casos de Uso Críticos (Nuevos)

- **CUC04:** Usuario ingresa "50 → M+ → 30 → M+ → MR" → Sistema muestra "80.00".
- **CUC05:** Usuario ingresa "200 → %" → Sistema calcula el 1% (2.00) y permite operar (ej: 200 + 10% = 220).

- **CUC06:** Usuario presiona "MC" → Sistema restablece memoria a 0.00.

## 12.5 Riesgos y Mitigación (Añadidos)

Riesgo	Mitigación
<b>Confusión con el botón %</b>	Incluir un tutorial interactivo en la primera ejecución.
<b>Memoria no visible para el usuario</b>	Mostrar un indicador en la pantalla principal (ej: "M=5.00").

## 14.3 Flujo de Datos (Actualizado)

### 1. Uso de Memoria:

- Al presionar "M+", el valor actual se envía a la capa de lógica para actualizar la memoria.
- "MR" recupera el valor desde la capa de datos y lo muestra en la GUI.

## Integración Técnica

- **En la GUI (Java Swing):**
  - Nuevos botones en JPanel inferior con ActionListener vinculado a Memoria.java.
  - Actualización dinámica del indicador de memoria en el JTextField.
- **En las Pruebas (JUnit):**
  - Casos de prueba para verificar:
    - Acumulación correcta (ej: assertEquals(15.0, memoria.recuperar()) tras M+=10 y M+=5).
    - Restablecimiento con MC (assertEquals(0.0, memoria.recuperar())).