

Investigación Bienes, Estructuración y Preparación

Nicolás Ramírez Cerón

Informe final de prácticas académicas para optar al título de Ingeniero Mecánico

Asesores

PhD. Andrés Felipe Agudelo Santamaría, Profesor Departamento Ing. Mecánica

Mauricio Cerquera Gaitán, Jefe Sección Servicios Etapa Avanzada Cobranza

Universidad de Antioquia

Facultad de Ingeniería

Departamento de Ingeniería Mecánica

Medellín

2025

Cita	Ramirez Ceron, 2025 [1]
Referencia Estilo IEEE (2020)	[1] Ramírez Cerón, N. "Investigación Bienes, Estructuración y Preparación", Semestre de Industria, Ingeniería Mecánica, Universidad de Antioquia, Medellin, 2025.







Repositorio Institucional: http://bibliotecadigital.udea.edu.co

Universidad de Antioquia - www.udea.edu.co

Rector: John Jairo Arboleda.

Decano Julio César Saldarriaga Molina.

Jefe departamento: Pedro León Simancas.

El contenido de esta obra corresponde al derecho de expresión de los autores y no compromete el pensamiento institucional de la Universidad de Antioquia, ni desata su responsabilidad frente a terceros. Los autores asumen la responsabilidad por los derechos de autor y conexos.

TABLA DE CONTENIDO

RESUMEN 8

ABSTRACT 9

I. Introducción 10

II. PLANTEAMIENTO DEL PROBLEMA 11

III. JUSTIFICACIÓN 12

IV. OBJETIVOS 13

A. Objetivo general 13

B. Objetivos específicos 13

V. Marco Teórico 15

A. Cobro jurídico y proceso prejurídico 15

B. Web scraping y automatización de extracción de datos 15

C. DOM 16

D. Integración de APIs para comunicación con entidades 16

E. Bases de datos y gestión de la información 16

F. Bizagi Modeler 16

G. Prioridad MoSCoW 17

H. Desarrollo de interfases gráficas con Tkinter 17

I. Orquestación de procesos y automatización de tareas 18

J. Transformación digital en procesos de cobranza 18

VI. Metodología 19
A. Fase 1: Análisis de requerimientos 19
1) Contexto Operativo: 19
2) Diagnóstico: 20
B. Fase 2: Diseño de la solución 22
1) El nuevo proceso de IB: 23
C. Fase 3: Desarrollo e implementación 24
1) Implementación de las funciones del orquestador. 25
2) Implementación de los webscrapers. 29
3) Implementación de la interfase gráfica. 36
D. Fase 4: Pruebas y validación 41
E. Fase 5: Despliegue y capacitación 42
VII. Resultados 44
VIII. Conclusiones 46
REFERENCIAS 47

LISTA DE TABLAS

I Clasificación de requerimientos *MoSCoW* 22

LISTA DE FIGURAS

1 Flujo de trabajo del colaborador. 21
2 Proceso de IB 24
3 Estructura general de la interfase gráfica. 41
4 Impactos sobre el proceso de IB EyP. 45

abreviaturas, acrónimos y siglas

LZ	Landing-Zone: Forma en la que coloquialmente se refieren los colaboradores a la base de datos del banco la cual está construida sobre la infraestructura Apache Hadoop.
Repizado	Es un deudor que anteriormente tuvo un proceso de cobro que logró pagar, reestructurar o re financiar, pero ahora cae de nuevo en impago.
SNR	Super Intendencia de Notariado y Registro.
API	Application Programming Interfaces.
IB	Investigación de bienes.
EYP	Estructuración y Preparación
DOM	Document Object Model.
XPATH	XML Path Language.
HTML	HyperText Markup Language.
SQL	Structured Query Language.
NIT	Numero único de Identificación Tributaria.
RUNT	Registro Único Nacional de Tránsito

RESUMEN

La herramienta desarrollada, titulada **"Investigación de Bienes, Estructuración y Preparación"**, fue diseñada con el propósito de brindar soporte integral al proceso de investigación de bienes del área de cobranza del banco. Esta solución se construyó en Python utilizando la biblioteca Tkinter para la interfase gráfica, bajo un enfoque modular, y con la ejecución de funciones en hilos o subprocesos según convenga, lo que garantiza eficiencia y estabilidad operativa.

El aplicativo permite a los colaboradores gestionar de manera interna y automatizada las solicitudes de investigación de bienes, proceso que anteriormente era tercerizado, lo que ha significado una reducción considerable en los costos asociados. Las funciones encargadas de obtener la información de los deudores se implementaron como *web scrapers*, permitiendo consultar diversas fuentes oficiales en línea de forma automática y estructurada.

Adicionalmente, el proyecto reemplazó el uso de herramientas ofimáticas como Ms Excel y Access por un sistema de almacenamiento basado en bases de datos SQLite, lo que permitió consolidar, consultar y resguardar la información de manera más segura, consistente y escalable, generando mayor trazabilidad y control del proceso.

Además de la investigación de bienes, la herramienta implementa las políticas necesarias para evaluar la viabilidad de cobro judicial, apoyando la preparación de la demanda con base en criterios definidos por el banco.

Por último, la solución actúa como un orquestador de tareas ofimáticas repetitivas que forman parte del proceso, centralizando funciones y optimizando el flujo de trabajo de principio a fin. Esto se traduce en un aumento significativo de la productividad del equipo, una toma de decisiones más precisa y una gestión de la información alineada con las mejores prácticas tecnológicas.

Palabras clave **— Automatización, cobranza, investigación de bienes, web scraping, minería de datos, Python, Tkinter, SQLite, orquestador de procesos.**

ABSTRACT

The developed tool, titled **"Asset Investigation, Structuring and Preparation"**, was designed to provide comprehensive support to the asset investigation process within the bank's collections department. This solution was built in Python using the Tkinter library for the graphical user interface, following a modular design and executing key functions through multithreading or subprocesses, depending on the context, to ensure operational efficiency and stability.

The application enables internal and automated management of asset investigation requests, a process that was previously outsourced, resulting in significant cost savings for the bank. The functions responsible for retrieving debtor information were implemented as *web scrapers*, allowing automatic and structured search of official online sources.

Additionally, the project replaced the use of office tools such as Ms Excel and Access with a structured storage system based on SQLite databases. This change enhanced the security, consistency, and scalability of the information, providing better traceability and control of the process.

Beyond asset investigation, the tool also implements the necessary policies to evaluate the feasibility of judicial collection, supporting legal decision-making based on criteria defined by the bank.

Lastly, the application acts as an orchestrator of repetitive office tasks associated with the process, centralizing operations and optimizing the workflow from end to end. This translates into a significant increase in team productivity, more precise decision-making, and improved information management aligned with technological best practices.

Keywords **— Automation, debt collection, asset investigation, web scraping, Python, Tkinter, SQLite, process orchestration.**

I. INTRODUCCIÓN

El proceso de cobranza en el sector financiero involucra múltiples etapas que buscan garantizar la recuperación efectiva de la cartera vencida, protegiendo al mismo tiempo los intereses de la entidad y respetando la normatividad vigente. Dentro de este proceso, la **investigación de bienes** y la **estructuración jurídica de los deudores** representan una fase clave para definir si existen fundamentos que justifiquen el inicio de acciones legales de cobro.

En el marco de la práctica profesional desarrollada en el área de cobranza del Banco Bancolombia, se identificó una oportunidad de mejora relacionada con la tercerización de este proceso y la alta dependencia de tareas manuales, las cuales requerían el uso de hojas de cálculo, bases de datos Access y otros recursos ofimáticos. Esta situación generaba altos costos operativos, riesgo de errores humanos y baja trazabilidad.

Como respuesta a esta necesidad, se diseñó y desarrolló una herramienta en Python, titulada **"Investigación de Bienes, Estructuración y Preparación"**, orientada a automatizar integralmente este proceso. La solución permite extraer, enviar, recibir y estructurar información proveniente de distintas entidades públicas y privadas, aplicar políticas internas de decisión y consolidar los resultados de forma organizada y trazable. Todo ello, a través de una interfase gráfica intuitiva que centraliza y orquesta las tareas necesarias, liberando al colaborador de labores repetitivas.

Este informe documenta el proceso completo de análisis, diseño, desarrollo e implementación del aplicativo, así como los resultados obtenidos y las conclusiones derivadas de su uso real dentro del entorno laboral.

II. PLANTEAMIENTO DEL PROBLEMA

Previo al desarrollo del presente proyecto, el proceso de investigación de bienes, estructuración y preparación dentro del área de cobranza del banco era ejecutado manualmente mediante herramientas ofimáticas como Ms Excel y Access. Adicionalmente, las solicitudes eran gestionadas por un tercero, lo que implicaba una dependencia operativa, altos costos recurrentes y limitaciones en el control y trazabilidad de la información.

Este enfoque presentaba múltiples inconvenientes, como el riesgo de errores humanos, pérdida de información, tiempos prolongados de respuesta y dificultades para integrar los resultados con otros procesos internos, como la preparación de la demanda judicial. Ante esta situación, surgió la necesidad de diseñar una solución interna que permitiera automatizar y centralizar dicho proceso, con el fin de optimizar los recursos y mejorar la calidad del servicio.

III. JUSTIFICACIÓN

La decisión del banco de internalizar el proceso de investigación de bienes responde a la necesidad de reducir costos, eliminar la dependencia de aliados externos y fortalecer el control sobre los procesos críticos de cobranza.

Al desarrollar una herramienta propia, el equipo puede gestionar las solicitudes de manera más ágil y precisa, garantizando la seguridad de la información y permitiendo su integración con otros sistemas institucionales. Además, la automatización del proceso libera a los colaboradores de tareas repetitivas, permitiéndoles enfocarse en actividades de mayor valor estratégico para la organización.

IV. OBJETIVOS

A. Objetivo general

Desarrollar una herramienta en Python, titulada **"Investigación de Bienes, Estructuración y Preparación"**, que automatice la gestión de solicitudes de investigación de bienes en el área de cobranza del banco, incluyendo la extracción de información, la evaluación de políticas de cobro y la organización estructurada de resultados, mediante técnicas de *web scraping*, uso de bases de datos SQL y una interfase gráfica interactiva.

B. Objetivos específicos

- Automatizar la extracción de solicitudes de investigación de bienes mediante técnicas de *web scraping* y consultas a bases de datos, permitiendo la identificación, clasificación y procesamiento de los NIT de los deudores.
- Implementar un sistema de consulta eficiente a entidades externas, integrando herramientas de automatización para el envío y recepción de información relevante en el proceso de investigación de bienes.
- Incorporar políticas de decisión que permitan determinar la viabilidad de iniciar cobro judicial, facilitando la estructuración y preparación de la demanda conforme a los lineamientos establecidos por el banco.
- Desarrollar una interfase gráfica intuitiva utilizando Tkinter, que permita al colaborador ejecutar los módulos de forma independiente o coordinada, con soporte para procesos en paralelo mediante *multithreading* o subprocesos.
- Reemplazar el uso de herramientas ofimáticas (Excel, Access) en la medida de lo posible por bases de datos SQLite para mejorar la seguridad, trazabilidad y escalabilidad en el manejo de la información.
- Validar la herramienta mediante pruebas de funcionamiento y retroalimentación del

usuario, asegurando la precisión de los datos, la eficiencia del flujo de trabajo y la facilidad de adopción en el entorno laboral.

V. MARCO TEÓRICO

La automatización de procesos mediante herramientas digitales se ha consolidado como una estrategia clave para optimizar la eficiencia operativa en el sector financiero. En este proyecto, se integran tecnologías de desarrollo de software, automatización de tareas y gestión de datos, para mejorar significativamente el proceso de investigación de bienes, una etapa crítica en la operación de cobranza.

A. Cobro jurídico y proceso prejurídico

El proceso de cobranza en entidades financieras se divide generalmente en etapas extrajudiciales (prejurídicas) y judiciales. En la fase prejurídica, se busca recuperar la cartera vencida a través de medios internos, sin necesidad de acudir a instancias legales. Una parte fundamental de este proceso es la **investigación de bienes**, que permite identificar los activos de los deudores con el fin de establecer si existen garantías suficientes para iniciar un proceso de cobro judicial [1, 2].

El proceso de *estructuración y preparación* de la demanda consiste en evaluar si, con base en los bienes identificados, se cumplen las condiciones para que el banco proceda jurídicamente. Automatizar esta tarea reduce el tiempo de respuesta y mejora la toma de decisiones, aportando eficiencia y trazabilidad al área de cobranza.

B. Web scraping y automatización de extracción de datos

El *web scraping* es una técnica que permite extraer información de páginas web de forma automatizada mediante *scripts* programados en lenguajes como Python. Librerías como `Selenium` y `BeautifulSoup` permiten interactuar con sitios web dinámicos, simular acciones del usuario y recopilar datos de manera estructurada [3, 4, 5].

En el contexto de este proyecto, el *web scraping* es fundamental para consultar fuentes oficiales como la Superintendencia de Notariado y Registro, el RUNT y la Cámara de Comercio, desde donde se extrae información clave sobre los bienes registrados a nombre de los deudores del banco.

C. DOM

El **DOM** (*Document Object Model*) es una interfase de programación para documentos HTML y XML. Representa la estructura de un documento como un árbol de nodos, donde cada nodo corresponde a una parte del documento (elementos, atributos, texto, etc.). El DOM permite a los lenguajes de programación acceder y modificar dinámicamente la estructura, el estilo y el contenido de los documentos web.

En el contexto de la automatización con Selenium, interactuar con el DOM es fundamental, ya que todas las acciones (búsqueda de elementos, clics, extracción de datos) se realizan sobre los nodos del DOM.

D. Integración de APIs para comunicación con entidades

Las **APIs** (*Application Programming Interfaces*) son interfases que permiten la comunicación entre sistemas de manera estructurada y segura. Su uso en entornos bancarios agiliza el intercambio de datos con plataformas externas. En este proyecto, las APIs permiten gestionar solicitudes hacia entidades aliadas y recibir respuestas en tiempo real, reemplazando flujos manuales e incrementando la confiabilidad del proceso [6, 7].

E. Bases de datos y gestión de la información

Una base de datos es un sistema organizado que permite almacenar, recuperar y manipular información de forma eficiente. En este caso, se hace uso de `SQLite`, una base de datos ligera y embebida, ideal para aplicaciones de escritorio. La migración de herramientas ofimáticas como Excel y Access hacia una base de datos estructurada mejora la consistencia, seguridad y trazabilidad de la información [8].

F. Bizagi Modeler

Bizagi Modeler es una herramienta de modelado de procesos de negocio que permite a las organizaciones diagramar, documentar y simular procesos de manera gráfica utilizando la notación estándar BPMN (*Business Process Model and Notation*). Esta aplicación, disponi-

ble en versión de escritorio de forma gratuita, facilita la colaboración entre equipos mediante funcionalidades para modelar, documentar, simular, publicar y compartir procesos. Los modelos creados pueden almacenarse localmente o en la nube, y la herramienta ofrece opciones avanzadas para publicar documentación en diversos formatos (Word, PDF, HTML, Wiki) e interoperar con otros sistemas mediante la exportación a formatos como Visio, XPDL o BPMN. Bizagi Modeler contribuye a la estandarización, trazabilidad y mejora continua de los procesos organizacionales, permitiendo una gestión eficiente del conocimiento y la gobernanza de los procesos empresariales [9].

G. Prioridad MoSCoW

La priorización MoSCoW es una técnica ampliamente utilizada en la gestión de requerimientos de proyectos de software y productos, que permite clasificar las funcionalidades según su importancia y urgencia para el éxito del proyecto. El término MoSCoW es un acrónimo de las categorías **Must have** (debe tenerse), **Should have** (debería tenerse), **Could have** (podría tenerse) y **Won't have** (no se tendrá en esta entrega), lo que facilita la toma de decisiones y la alineación de expectativas entre los equipos de desarrollo y los interesados en futuras iteraciones.

Esta técnica fue desarrollada por Dai Clegg en Oracle en la década de 1990 y posteriormente adoptada en metodologías ágiles y entornos de gestión de productos digitales. El uso de MoSCoW facilita la comunicación entre los equipos y los interesados, asegurando que los elementos críticos sean abordados primero y que las expectativas estén alineadas con los recursos y el cronograma del proyecto [10, 11, 12, 13].

H. Desarrollo de interfases gráficas con Tkinter

`Tkinter` es el módulo estándar de Python para el desarrollo de interfases gráficas (GUI). Permite construir aplicaciones de escritorio interactivas y personalizadas, facilitando la experiencia del usuario en entornos no técnicos. La solución implementada hace uso de Tkinter junto con `multithreading`, lo que permite ejecutar múltiples procesos de forma

simultánea sin afectar la usabilidad de la interfaz [14, 15].

I. Orquestación de procesos y automatización de tareas

La **orquestación de procesos** consiste en coordinar la ejecución de múltiples tareas dentro de un flujo de trabajo, optimizando el uso de recursos y tiempos de respuesta. En este caso, el aplicativo actúa como una plataforma central desde la cual el colaborador puede ejecutar consultas, recibir resultados, aplicar políticas de decisión y generar salidas organizadas [16, 17], eliminando tareas manuales repetitivas que antes requerían intervención directa mediante hojas de cálculo.

J. Transformación digital en procesos de cobranza

La transformación digital en instituciones financieras busca modernizar procesos críticos mediante la incorporación de tecnologías que mejoren la eficiencia, seguridad y capacidad de análisis de la información. Automatizar la gestión de investigaciones de bienes es un paso hacia esa transformación, ya que permite integrar procesos, reducir costos operativos y facilitar decisiones fundamentadas en datos, lo cual es vital en la gestión de cartera y recuperación de activos [18, 19].

VI. METODOLOGÍA

El desarrollo de la herramienta se abordó mediante una metodología aplicada, orientada a la solución de una necesidad real identificada en el proceso de investigación de bienes del área de cobranza del banco. El enfoque fue iterativo, combinando análisis de procesos con diseño técnico, desarrollo ágil, validación funcional y despliegue progresivo.

El proyecto se estructuró en cinco fases principales:

A. Fase 1: Análisis de requerimientos

1) Contexto Operativo: El proceso manual de Investigación de Bienes, Estructuración y Preparación presentaba una arquitectura operativa fragmentada caracterizada por:

- La administración de los clientes en situación de mora se realizaba de forma manual, requiriendo que el colaborador accediera diariamente a un portal específico para completar los campos necesarios y seleccionar las variables pertinentes. Este procedimiento permitía la generación y posterior descarga de un archivo Excel con el reporte actualizado de los deudores, el cual servía como base para la gestión operativa subsecuente.
- Las solicitudes de Investigación de Bienes se gestionaban manualmente mediante la descarga periódica de hojas de cálculo generadas desde Bizagi Modeler, las cuales eran gestionadas manualmente por el colaborador para su procesamiento. Adicionalmente, las regionales podían remitir sus solicitudes directamente al colaborador encargado a través de correo electrónico. Esta multiplicidad de canales y la ausencia de un formato estandarizado generaban una alta dispersión en la información recibida. Como resultado, los datos consolidados presentaban frecuentes inconsistencias atribuibles a errores humanos, tales como diferencias en el uso de acentos, palabras mal escritas o campos incompletos, lo que dificultaba la gestión de la información y aumentaba el riesgo de reprocesos.
- El flujo de trabajo involucraba tres roles diferentes (Encargado de IB, Torre de Control (asignación) y gestor documental) con interacciones asincrónicas.

- Las investigaciones de bienes eran realizadas por un proveedor tercero que tarda en responder 5+ días hábiles las solicitudes gestionadas por el colaborador.
- El 78 % del tiempo de proceso se dedicaba a tareas repetitivas: copiado de datos entre sistemas (32 %), validación manual de información (27 %) y seguimiento de estados (19 %).

A pesar de la dispersión operativa, los líderes del proceso tenían claridad sobre los alcances y funcionalidades esperadas del aplicativo, definiendo como prioridad la centralización de la gestión y la reducción de tareas manuales. Durante el diagnóstico, se identificó que el orquestador fue inicialmente propuesto para dar soporte exclusivo al proceso de IB. Sin embargo, debido a que los procesos de estructuración y preparación eran gestionados/asignados por el mismo colaborador encargado de IB, y además dependían directamente de la información consolidada en este módulo, se optó por ampliar el alcance del orquestador; observe el diagrama incluido en la figura 1.

2) Diagnóstico: Se realizó un diagnóstico detallado del proceso actual de investigación de bienes mediante entrevistas y sesiones de observación con los colaboradores del área. Se identificaron los puntos críticos del flujo de trabajo manual, incluyendo:

- Altos tiempos de procesamiento por dependencia de herramientas ofimáticas (Excel y Access).
- Riesgos de pérdida de información y errores humanos.
- Dificultades para consolidar y trazar resultados.
- Dependencia de un proveedor externo para realizar las investigaciones.

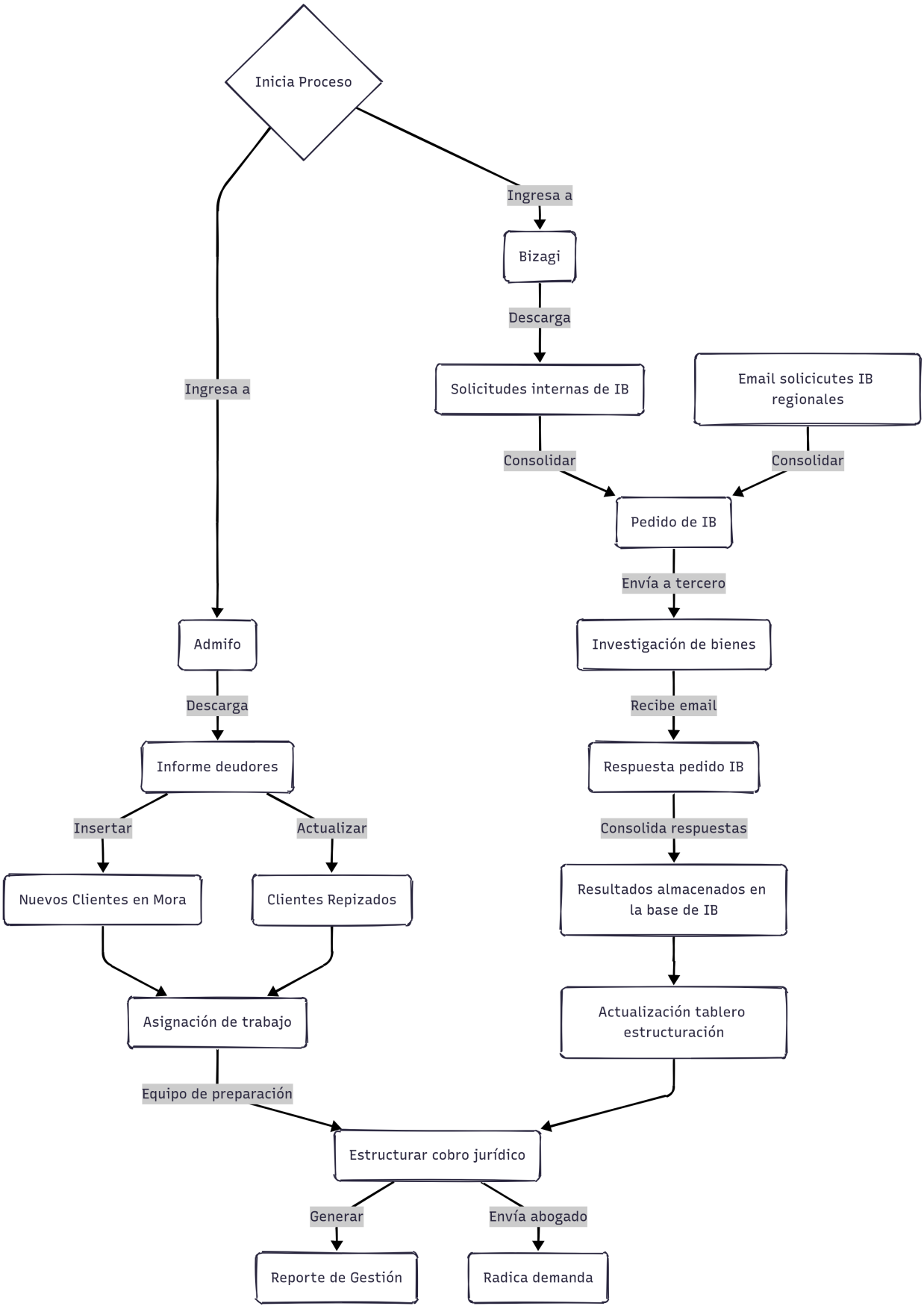


Fig. 1. Flujo de trabajo del colaborador.

Se realizó un análisis MoSCoW para priorizar y clasificar los elementos clave del proyecto. Ver tabla I

Categoría	**Cantidad de Procesos**	**Prioridad MoSCoW**
Gestión de Clientes en Mora	9	Should
Gestión de solicitudes de IB	7	Must
Integrar Consultas Salarios, Propiedades —Vehículos y Cámara de Comercio	12	Must
Aplicación de Políticas de cobro	5	Must
Gestor Documental	8	Should
Asignación de Trabajo	6	Could
Reportes	8	Won't

TABLA I
CLASIFICACIÓN DE REQUERIMIENTOS *MoSCoW*

Como resultado, se establecieron los requerimientos funcionales y técnicos de la herramienta.

B. Fase 2: Diseño de la solución

Con base en los requerimientos, se diseñó la arquitectura de la aplicación en Python, siguiendo un enfoque modular. Se definieron los siguientes componentes:

- **funciones de torre de control (Adminfo):** proporciona al colaborador las herramientas necesarias para gestionar la distribución del trabajo entre el equipo de prejurídicos, gestionando la información de los nuevos deudores y actualizando el estado de las obligaciones de los clientes en mora.
- **funciones de gestión solicitudes IB (Bizagi):** encargado de recopilar solicitudes y datos de deudores desde fuentes internas mediante *scripts* de python.
- **Módulo de consulta externa:** Utiliza *web scrapers* y APIs para recopilar datos tanto de entidades gubernamentales como no gubernamentales, proporcionando información

detallada sobre propiedades, vehículos, salarios y empresas registradas a nombre de los clientes en mora. Este se dividió en dos paneles distintos, para cada instancia, envío de la solicitud de información y recepción de las respuestas.

- **Funciones de estructuración prejurídica:** analiza los resultados obtenidos en el modulo de consulta externa y aplica las políticas del banco para determinar la viabilidad del cobro judicial.
- **Módulo Análisis Inteligente de Documentos:** Se debe hacer uso de las capacidades del banco para analizar los certificados de libertad y tradición con herramientas de IA y determinar si los folios son productivos. Lo que implica que se debe construir *scripts* que permitan enviar los documentos al área de análisis de documentos y consultar las respuestas en una base de datos interna, además de insertar los resultados en la base de datos de IB.
- **Interfase gráfica:** construida en Tkinter con soporte para *multithreading*, permitiendo interacción fluida y ejecución paralela de tareas.
- **Base de datos local en SQLite:** para consolidar, organizar y consultar los resultados de manera segura y eficiente.

1) El nuevo proceso de IB: La Figura 2 muestra el esquema del proceso de investigación de bienes semi-automatizado que se estableció en colaboración con el encargado y los líderes del proceso. Este diagrama sintetiza las interacciones entre diversos módulos creados o funciones del orquestador, abarcando desde la recepción de solicitudes internas hasta la integración de resultados, implementación de políticas de decisión y consolidación de los resultados de dichas políticas.

El diagrama de la Figura 2 también refleja el reemplazo del proceso externo de IB por un flujo estructurado, centralizado y ejecutado desde la herramienta orquestadora.

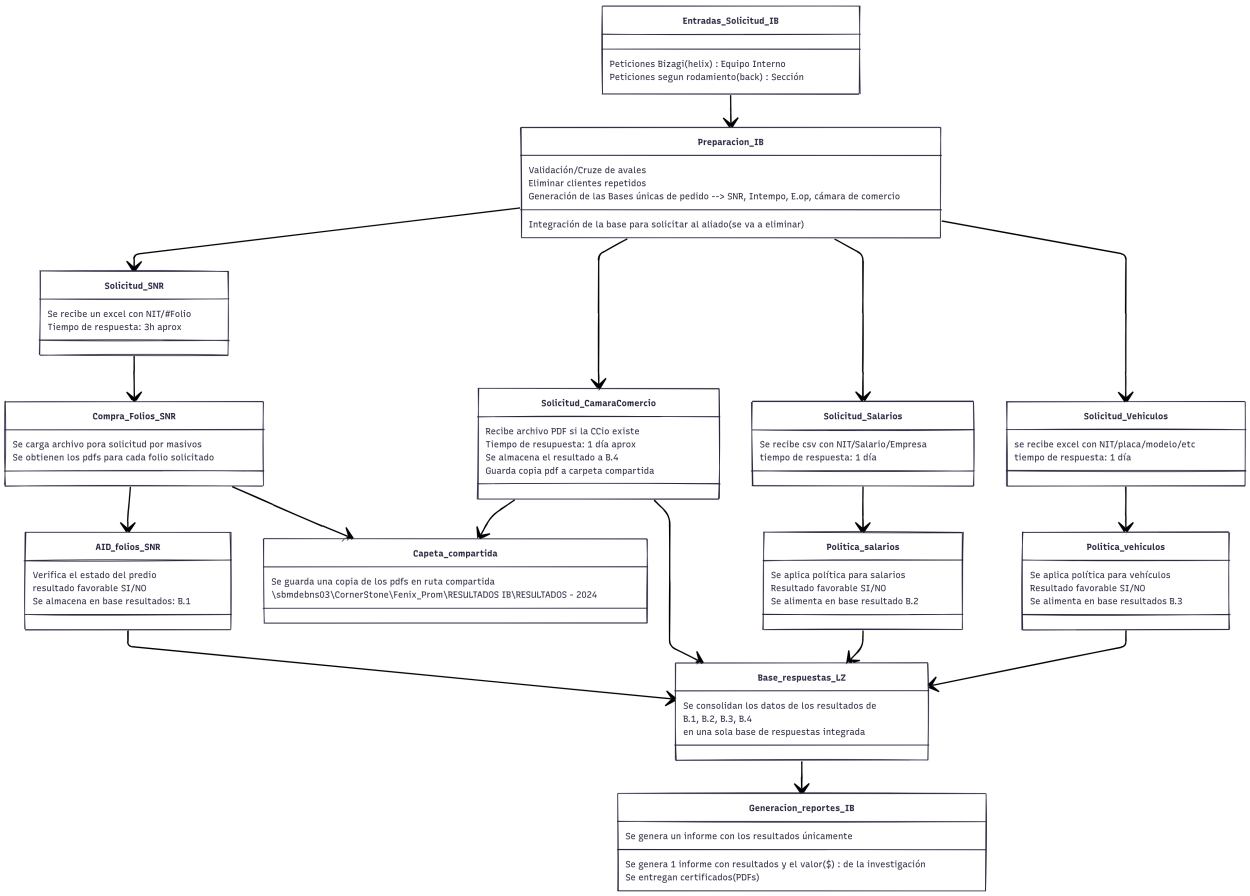


Fig. 2. Proceso de IB

C. Fase 3: Desarrollo e implementación

Se desarrollaron los módulos de forma incremental, iniciando con la automatización de extracción, y continuando con la integración con entidades externas. Las funcionalidades se integraron progresivamente en una interfase amigable para el usuario final.

Durante esta fase, se adoptó una estrategia basada en el desarrollo y prueba de funciones dentro de cuadernos interactivos en formato `.ipynb` (Jupyter Notebooks), lo que permitió una validación constante del comportamiento de cada componente frente a los resultados del proceso manual. Esto facilitó la depuración de errores y la comparación directa de salidas, asegurando la consistencia con las expectativas del equipo operativo.

Adicionalmente, se mantuvo un esquema de entrega continua al colaborador responsable del proceso, proporcionándole versiones preliminares funcionales del aplicativo con los módulos implementados hasta el momento. Esto permitió obtener retroalimentación temprana y realizar ajustes de manera ágil, fortaleciendo el proceso de co-construcción de la herramienta.

Las tecnologías y librerías utilizadas en esta fase incluyeron:

- Python 3.8
- Jupyter Notebooks (`.ipynb`) para desarrollo y validación de funciones
- Tkinter para el desarrollo de la interfase gráfica
- Selenium y Requests para *scraping* y automatización web
- SQLite3 como sistema de almacenamiento local
- *Multithreading* mediante el módulo `threading` de Python

Cada módulo se validó individualmente antes de integrarse en el sistema general. A continuación se detallarán las características fundamentales y ciertas prácticas empleadas para la implementación de cada uno de los módulos esenciales del orquestador.

1) Implementación de las funciones del orquestador. Las funciones *no-webscraper* del orquestador pueden agruparse en las siguientes categorías, cada una con su propósito y ejemplos de uso:

1. **Gestión de Archivos y Descargas**

   Estas funciones aseguran la correcta manipulación de archivos generados o descargados por los procesos automáticos. Incluyen verificación de descargas, renombrado, descompresión y limpieza de archivos temporales.

```
# Verifica que un archivo se descargue completamente
def verificar_descarga(ruta_descarga, filename, tiempo_espera=60):
```

```
        # ...existing code...
        if os.path.exists(archivo_descargado):
            return True
    # ...existing code...

    # Elimina archivos temporales .crdownload antes de iniciar un nuevo proceso
    def eliminar_archivos_crdownload():
        # ...existing code...
        if archivo.endswith(".crdownload"):
            os.remove(ruta_archivo)
    # ...existing code...
```

2. **Procesamiento y Normalización de Datos**

   Permiten limpiar, transformar y estandarizar datos provenientes de archivos Excel o bases de datos, facilitando su integración y análisis.

```
    # Normaliza nombres de columnas para compatibilidad con bases de datos
    def normalizar_claves_columnas(df):
        df.columns = (
            df.columns.str.upper()
            .str.replace(" ", "_", regex=True)
            .str.normalize('NFKD')
            .str.encode('ascii', errors='ignore').str.decode('utf-8')
        )
        return df

    # Corrige entradas de folios mal formateadas
    def corregir_folios(valor):
        valor = re.sub(r'\s+y\s+', ';', valor)
        valor = re.sub(r'\s*,\s*', ';', valor)
        return valor
```

3. **Gestión y Sincronización de Bases de Datos**

Manejan la persistencia, actualización y consulta de pedidos y resultados en bases de datos locales (`sqlite3`), permitiendo el seguimiento y trazabilidad de los procesos.

```
    # Guarda un nuevo pedido en la base de datos
    def guardar_pedido(codigo, entidad, solicitante, estado, fecha):
        conn = sqlite3.connect('pedidos.db')
        cursor = conn.cursor()
        cursor.execute('''
            INSERT INTO pedidos (codigo, entidad, solicitante, estado, fecha)
            VALUES (?, ?, ?, ?, ?)
        ''', (codigo, entidad, solicitante, estado, fecha))
        conn.commit()
        conn.close()

    # Cambia el estado de un pedido existente
    def cambiar_estado_pedido(codigo, nuevo_estado):
        # ...existing code...
        cursor.execute('UPDATE pedidos SET estado = ?, fecha = ? WHERE codigo = ?', (
            ↪ nuevo_estado, fecha_actual, codigo))
        # ...existing code...
```

4. **Integración y Sincronización con Bases de Datos Externas**

   Ejecutan consultas SQL sobre bases empresariales (por ejemplo, Access, Impala) para obtener información relevante y sincronizar datos entre sistemas.

```
    # Consulta la LZ usando pyodbc y retorna un DataFrame
    def consulta_LZ(query):
        cn=pyodbc.connect("DSN=impala-prod", autocommit=True)
        df=pd.read_sql(query,cn)
        cn.close()
        return df
```

5. **Automatización de Procesos de Negocio**

   Automatizan tareas administrativas y de integración de datos, como la actualización de archivos Excel, organización de documentos, y preparación de formatos para solicitudes.

```python
# Organiza archivos PDF en carpetas por NIT segn un Excel
def organizador_folios(excel, folder):
    df = pd.read_excel(excel)
    matricula_to_nit = dict(zip(df['MATRICULA'], df['NIT']))
    for filename in os.listdir(folder):
        if filename.endswith('.pdf'):
            matricula = filename.split('-')[1]
            nit = matricula_to_nit.get(int(matricula))
            if nit:
                nit_directory = os.path.join(folder, str(nit))
                os.makedirs(nit_directory, exist_ok=True)
                shutil.move(os.path.join(folder, filename), os.path.join(
                    ↪ nit_directory, filename))
```

6. **Interfase de Usuario y Soporte a la Visualización**

Facilitan la interacción con el usuario, mostrando mensajes, logs y resultados en consola o interfases gráficas (Tkinter).

```python
# Redirige la salida de consola a un widget de Tkinter
class ConsoleRedirector:
    def __init__(self, widget):
        self.widget = widget
    def write(self, message):
        self.widget.insert(tk.END, message)
        self.widget.see(tk.END)
        self.widget.update_idletasks()
```

7. **Envío de Correos y Notificaciones**

Automatiza el envío de correos electrónicos con archivos adjuntos, integrándose con Outlook mediante COM.

```python
#  Enva  un email con archivo adjunto usando Outlook
def send_email_ccio(archivo):
    outlook = win32.Dispatch('outlook.application')
```

```
        mail = outlook.CreateItem(0)
        mail.Subject = "Asunto"
        mail.To = "destinatario@dominio.com"
        mail.Attachments.Add(archivo)
        mail.Send()
```

8. **Aplicación de Políticas de Negocio**

   Analizan los resultados obtenidos y aplican reglas de negocio para determinar el estado o resultado de cada caso, actualizando la base de datos principal.

```
# Aplica polticas para clasificar resultados de vehculos
def politicas_Vehiculos(resultados_pedido):
    for index, row in resultados_pedido.iterrows():
        antiguedad = datetime.datetime.now().year - int(row["Modelo Vehculo"])
        if antiguedad <= 6:
            resultado_politica = "Positiva"
        else:
            resultado_politica = "Negativa"
        resultados_pedido.at[index, "Resultado_IB"] = resultado_politica
```

9. **Utilidades y Soporte a Multithreading**

   Permite ejecutar funciones en hilos independientes para mejorar la experiencia de usuario y la eficiencia de la aplicación.

```
# Ejecuta una funcin en un hilo separado
def hilos(funcion, boton=None):
    def wrapper():
        funcion()
    threading.Thread(target=wrapper).start()
    if boton:
        boton.config(bg='green')
```

2) Implementación de los webscrapers. Los *Scripts* están implementados siguiendo una arquitectura procedimental, donde cada función encapsula un flujo de automatización web

específico para interactuar con diferentes portales y sistemas externos. La estructura general presentada a continuación se desarrolló basándose en lel libro de Chapagain [20] además de los artíciculos de Gunawan, Rahmatulloh y Darmawan [21].

1. **Inicialización del navegador**

   Cada función que automatiza un portal inicia una instancia de Edge, configurando opciones y preferencias. Ejemplo:

```
    # ...existing code...
    service = Service('msedgedriver.exe')
    options = Options()
    options.add_argument("--allow-running-insecure-content")
    options.add_argument("user-agent=Mozilla/5.0 ...")
    driver = webdriver.Edge(service=service, options=options)
    driver.maximize_window()
    driver.get(config['url_SNR'])
    # ...existing code...
```

2. **Autenticación y navegación**

   Se automatiza el login y la navegación por menús usando selectores robustos y esperas explícitas:

```
    # ...existing code...
    usuario = driver.find_element(By.ID, "usuario")
    usuario.send_keys(config['Usuario_Adminfo'])
    clave = driver.find_element(By.ID, "clave")
    clave.send_keys(config['Contrasea_Adminfo'])
    entrar = WebDriverWait(driver,40).until(
        EC.presence_of_element_located((By.XPATH, "..."))
    )
    entrar.click()
    # ...existing code...
```

3. Interacción con formularios y tablas

Los *scripts* rellenan formularios, seleccionan opciones y recorren tablas para encontrar datos o realizar acciones. Por ejemplo, en las funciones de descarga de resultados es normal recorrer una tabla para buscar un archivo específico:

```
    # ...existing code...
    rows = WebDriverWait(driver, 40).until(
        EC.visibility_of_all_elements_located((By.CSS_SELECTOR,
        "table tbody tr"))
    )
    coincidencia = False
    for row in rows[1:]:
        cells = row.find_elements(By.TAG_NAME, "td")
        if len(cells) > 1:
            try:
                link = cells[1].find_element(By.TAG_NAME, "a")
                name = link.text.strip()
                if numero_pedido in name:
                    print_date(f"Click en: {name}")
                    driver.execute_script("arguments[0].scrollIntoView(true);", link)
                    ActionChains(driver).move_to_element(link).click().perform()
                    coincidencia = True
                    break
            except Exception as e:
                print(f"      Error al intentar hacer click: {e}")
                continue
    # ...existing code...
```

También se usan formularios y botones:

```
    # ...existing code...
    file_input = WebDriverWait(driver, 40).until(
        EC.presence_of_element_located((By.XPATH, "...input[@type='file']"))
    )
    file_input.send_keys(file_path)
    boton_aceptar = driver.find_element(By.XPATH, ".../button")
```

```
    boton_aceptar.click()
    # ...existing code...
```

4. **Descarga y manejo de archivos**

Se verifica la descarga de archivos y se procesan según el flujo de negocio:

```
    # ...existing code...
    def verificar_descarga(ruta_descarga, filename, tiempo_espera=60): #Verifica que
        ↪  los archivos se descarguen completamnete
        tiempo_inicial = time.time()

        while time.time() - tiempo_inicial < tiempo_espera:
            # Esperar 15 segundos para que la descarga comience correctamente
            time.sleep(5)

            # Verificar si existen archivos temporales
            archivos_temporales = [f for f in os.listdir(ruta_descarga) if f.endswith
                ↪ ('.crdownload') or f.endswith('.part')]
            if not archivos_temporales:
                # Verificar si el archivo descargado existe
                archivo_descargado = os.path.join(ruta_descarga, filename)
                if os.path.exists(archivo_descargado):
                    return True

        return False
    # ...existing code...
```

5. **Procesamiento de resultados**

Una vez descargados, los archivos se leen y los datos se integran a sistemas internos:

```
    # ...existing code...
    df_resultado = pd.read_excel(nueva_ruta)
    for _, row in df_resultado.iterrows():
        nit = row["PARAMETROS"]
        folio = f"{row['OFICINA']}-{row['MATRICULA']}"
```

```
        # ...actualizacin de base interna...
    # ...existing code...
```

6. **Sincronización y robustez**

   Se emplean esperas explícitas y manejo de excepciones para asegurar la robustez ante cambios en la web:

```
    # ...existing code...
    try:
        elemento = WebDriverWait(driver, 40).until(
            EC.presence_of_element_located((By.XPATH, "..."))
        )
    except Exception as e:
        print("Elemento no encontrado:", e)
    # ...existing code...
```

7. **Cierre de recursos**

   Al finalizar, se cierra el navegador para liberar recursos:

```
    # ...existing code...
    driver.quit()
    # ...existing code...
```

Otras consideraciones importantes sobre los *scrapers* desarrollados incluyen:

- **Manejo de autenticación y MFA:** Los *scrapers* implementan flujos para autenticación robusta, incluyendo el manejo de códigos de autenticación multifactor (MFA) mediante ventanas emergentes para interacción del usuario.
- **Sincronización y tolerancia a fallos:** Se utilizan esperas explícitas (`WebDriverWait`) y manejo de excepciones para adaptarse a la latencia de la red y cambios en la estructura de las páginas, mejorando la resiliencia ante fallos.

- **Automatización de descargas y procesamiento de archivos:** Los *scripts* no sólo descargan archivos, sino que también verifican su integridad, los renombran, descomprimen y mueven a rutas específicas para su posterior análisis.
- **Integración con sistemas internos:** Los resultados obtenidos se integran automáticamente a bases de datos locales (SQLite, Access) a archivos Excel, permitiendo la actualización y trazabilidad de la información.
- **Interacción con el usuario:** En puntos críticos (validación visual, ingreso de MFA, confirmaciones), los *scrapers* solicitan intervención del usuario mediante interfases gráficas ligeras (Tkinter).
- **Manejo de sesiones y *cookies*:** En algunos casos, se reutilizan *cookies* de sesión de Selenium en peticiones HTTP directas para descargar archivos que el navegador bloquea.
- **Mantenimiento y adaptabilidad:** El código está preparado para cambios frecuentes en la interfase de los portales, facilitando la actualización de selectores y flujos.
- **Seguridad y cumplimiento:** Se evita el almacenamiento de credenciales en texto plano y se promueve el uso de archivos de configuración externos y variables de entorno.
- **Registro y trazabilidad:** Se implementan funciones de *logging* con marcas de tiempo y mensajes claros para facilitar la auditoría y el diagnóstico de errores.
- **Escalabilidad y modularidad:** Cada *scraper* está encapsulado en funciones independientes, lo que permite su reutilización y facilita la integración en flujos más complejos.

Estrategias implementadas para lidiar con entornos dinámicos y entornos de usuario:

- **Esperas explícitas (`WebDriverWait`):** Se utiliza para esperar a que el botón esté presente en el DOM antes de intentar cualquier acción, evitando errores por elementos aún no cargados.

```
# ...existing code...
descargar_btn = WebDriverWait(driver, 40).until(
    EC.presence_of_element_located((By.ID, "object-detail-download-object-button"
        ↪ ))
)
# ...existing code...
```

- **Verificación de visibilidad:** Se asegura que el botón no sólo esté presente, sino también visible para el usuario.

```
# ...existing code...
WebDriverWait(driver, 10).until(
    EC.visibility_of(descargar_btn)
)
# ...existing code...
```

- **Verificación de clicabilidad:** Se comprueba que el botón sea interactuable, es decir, que no esté cubierto por *overlays* o deshabilitado.

```
# ...existing code...
WebDriverWait(driver, 10).until(
    EC.element_to_be_clickable((By.ID, "object-detail-download-object-button"))
)
# ...existing code...
```

- ***Scroll* hacia el elemento:** Se realiza un desplazamiento automático para asegurar que el botón esté dentro del área visible de la ventana del navegador.

```
# ...existing code...
driver.execute_script("arguments[0].scrollIntoView({block: 'center'});",
    ↪ descargar_btn)
# ...existing code...
```

- **Acción con `ActionChains`:** Se utiliza para mover el cursor y hacer clic de manera más robusta, especialmente útil cuando hay capas dinámicas o animaciones.

```
    # ...existing code...
    ActionChains(driver).move_to_element(descargar_btn).click().perform()
    # ...existing code...
```

- **Clic mediante JavaScript:** Como último recurso, si el clic convencional falla (por *overlays* o problemas de foco), se ejecuta un clic directamente vía JavaScript.

```
    # ...existing code...
    driver.execute_script("arguments[0].click();", descargar_btn)
    # ...existing code...
```

Estas técnicas combinadas maximizan la probabilidad de éxito en la interacción con elementos dinámicos o problemáticos en interfases web modernas.

3) Implementación de la interfase gráfica.

1. **Arquitectura General**

   La interfase gráfica está implementada en Python utilizando `tkinter` como *framework* principal, complementado con `ttk` para *widgets* avanzados y `PIL (Pillow)` para el manejo de imágenes. El diseño sigue una estructura modular basada en la clase `FishToolsApp`, que encapsula toda la lógica y los componentes visuales de la aplicación.

2. **Estructura de la Ventana Principal**

   - **Ventana raíz (`root`):** Es la ventana principal de la aplicación, configurada con título, icono y color de fondo.
   - **Panel de navegación izquierdo:** Un `Frame` vertical de color negro que contiene el logo institucional y los botones de navegación para acceder a los diferentes módulos.
   - **Área principal:** Un `Frame` que ocupa el resto de la ventana, donde se cargan dinámicamente los distintos paneles (*frames*) según el módulo seleccionado.

3. **Panel de Navegación Izquierdo**

   - **Logo institucional:** Imagen de la empresa, redimensionada y ubicada en la parte inferior del panel.
   - **Botones de módulos:** Cada botón representa un módulo funcional (ej. Adminfo, Bizagi, Envío/Recepción de Solicitudes IB, AID, Tools, Configuraciones). Los botones incluyen iconos personalizados y están alineados verticalmente.
   - **Gestión de eventos:** Al hacer clic en un botón, se invoca el método `mostrar_frame`, que oculta los paneles activos y muestra el correspondiente al módulo seleccionado.

4. **Área Principal y Paneles Dinámicos**

   El área principal contiene diferentes paneles (*frames*), cada uno asociado a un módulo funcional. Estos paneles se crean mediante métodos dedicados y se almacenan en un diccionario para su gestión dinámica.

   - **Panel de Inicio**
     - **Canvas central:** Muestra una imagen de bienvenida (mascota o logo del sistema), redimensionada dinámicamente según el tamaño de la ventana.
     - **Funcionalidad:** No contiene botones ni entradas, solo la imagen y el fondo blanco.
   - **Panel Adminfo**
     - **Botones de acción:** Incluye botones para descargar el Back de Adminfo, generar archivos de saldos, importar bases Access, generar reportes de clientes nuevos, llevar la base a la ruta compartida, y abrir el módulo de tablero y avalistas. Cada botón tiene un icono representativo.
     - **Tabla de gestión:** Una tabla (`ttk.Treeview`) muestra el estado de gestión de los usuarios, clientes y obligaciones, con opción de filtrar por tipo de gestión mediante un `Combobox`.

- **Consola de salida:** Un *widget* `ScrolledText` muestra *logs* y mensajes de las operaciones realizadas.
- **Botones de soporte:** Incluye botones para abrir la carpeta *Output*, limpiar archivos, y limpiar la consola.

- **Panel Bizagi**
  - **Carga de archivos:** Permite seleccionar un archivo Excel de solicitudes de investigación de bienes mediante un diálogo de archivos.
  - **Selección de solicitante:** Un *Combobox* permite elegir el tipo de solicitante (Interno, Regionales, o aliados).
  - **Ejecución de flujo:** Botón para ejecutar el flujo Bizagi con los parámetros seleccionados.
  - **Consola y soporte:** Consola de *logs* y botones para abrir la base IB y limpiar la consola.
- **Panel Envío de Solicitudes IB**
  - **Notebook de subpaneles:** Utiliza un `ttk.Notebook` para organizar subpaneles por entidad (SNR, Intempo, Enlace Operativo, Cámara de Comercio, etc.).
  - **Subpanel ALL:** Incluye botones para obtener solicitudes, rellenar formatos, y gestionar archivos de entrada. Muestra una tabla de archivos disponibles y un logo institucional.
  - **Subpaneles por entidad:** Cada subpanel permite seleccionar archivos, ejecutar funciones específicas (ej. recargar saldo, enviar solicitudes), y visualizar archivos disponibles en la carpeta Input.
  - **Consola de *logs*:** Cada subpanel tiene su propia consola de salida.
- **Panel Recepción de Solicitudes IB**
  - **Tabla de pedidos:** Muestra los pedidos pendientes en una tabla, permitiendo seleccionar uno para procesar.

  - **Panel de acciones:** Botones para descargar resultados de cada entidad (SNR, Intempo, Enlace, CCio), y para aplicar políticas manuales sobre archivos seleccionados.
  - **Procesamiento manual:** Permite seleccionar el tipo de política y el archivo a procesar, ejecutando la función correspondiente.
  - **Consola y soporte:** Consola de logs y botones para abrir bases y limpiar la consola.

- **Panel AID**
  - **Tabla de folios:** Muestra los folios y su estado en una tabla, con opción de filtrar por estado.
  - **Gestión de folios:** Permite seleccionar y enviar archivos PDF a la ruta AID, registrando el folio en la base de datos.
  - **Botones de soporte:** Incluye botones para refrescar la tabla, obtener resultados, y limpiar la consola.

- **Panel Tools**
  - **Selector de herramientas:** Un `Combobox` permite elegir entre herramientas utilitarias (consulta única SNR, organizador de folios, organizador de cámaras).
  - **Subpaneles dinámicos:** Cada herramienta tiene su propio subpanel con campos y botones específicos.
  - **Consola de logs:** Consola compartida para mostrar resultados y mensajes.

- **Panel Configuración**
  - **Visualización y edición de configuración:** Muestra todos los parámetros de configuración en campos editables, con opción de filtrar por categoría.
  - **Soporte de scroll:** Utiliza un `Canvas` y `Scrollbar` para manejar grandes cantidades de parámetros.

- **Botón de guardado:** Permite guardar los cambios realizados en la configuración.

5. **Características Avanzadas y Buenas Prácticas**

- ***Multithreading:*** Todas las funciones que pueden ser largas o bloqueantes se ejecutan en hilos separados mediante la función `hilos`, evitando que la interfase se congele.
- **Redirección de consola:** Se utiliza la clase `ConsoleRedirector` para redirigir la salida estándar (`print`) a las consolas de cada panel, mejorando la trazabilidad y experiencia de usuario.
- **Gestión de imágenes:** Todas las imágenes se cargan y redimensionan dinámicamente para adaptarse a los diferentes tamaños de *widgets* y resoluciones de pantalla.
- **Soporte para archivos y carpetas:** Los paneles incluyen botones para abrir carpetas relevantes (Input, Output, Resultados, AID) y seleccionar archivos mediante diálogos estándar.
- **Estilos personalizados:** Se definen estilos para tablas y *widgets*, mejorando la estética y la usabilidad.
- **Validación y *feedback*:** Se valida la selección de archivos y entradas antes de ejecutar funciones críticas, mostrando mensajes claros en la consola.
- **Modularidad y escalabilidad:** Cada panel y subpanel está implementado como un método independiente, facilitando la extensión y el mantenimiento de la interfase.

6. **Resumen**

La interfase gráfica está diseñada para ser intuitiva, robusta y modular, permitiendo la gestión eficiente de los procesos de investigación de bienes, integración con múltiples sistemas externos, y soporte a tareas administrativas y de negocio. Su arquitectura

facilita la escalabilidad y el mantenimiento, y su diseño visual mejora la experiencia de usuario y la trazabilidad de las operaciones. En la figura 3 puede observar el resumen y características generales de la interfase.

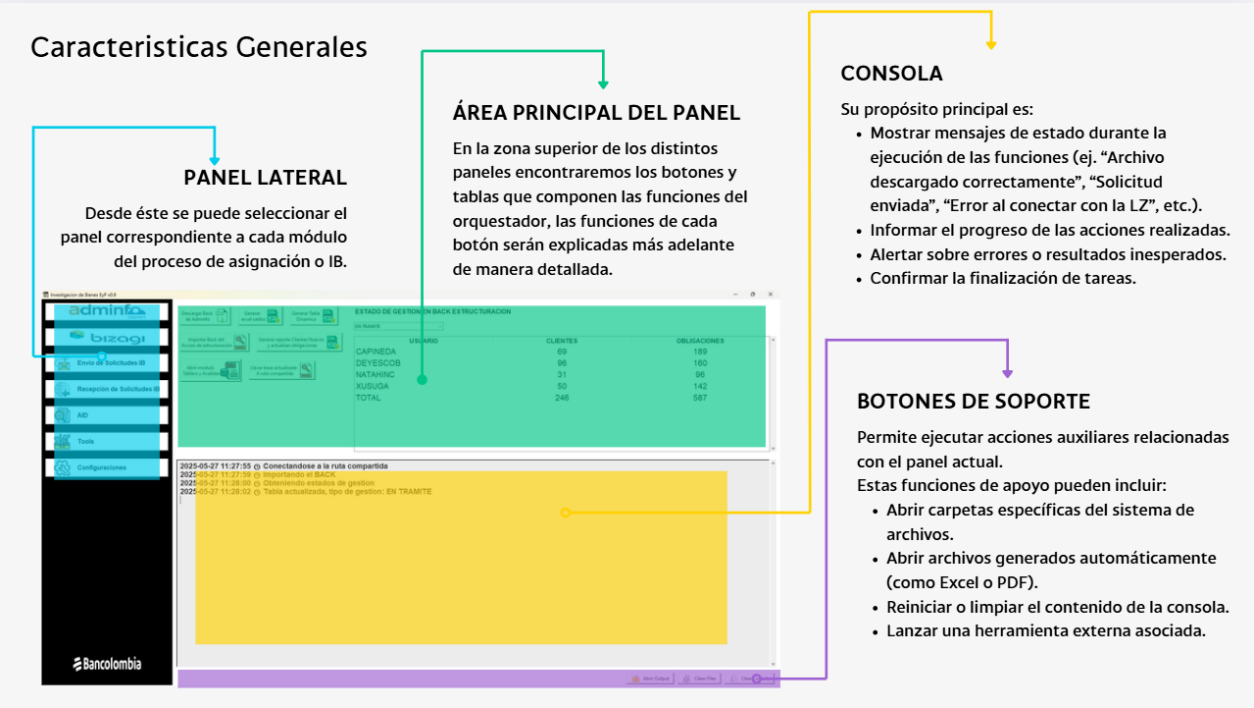


Fig. 3. Estructura general de la interfase gráfica.

D. Fase 4: Pruebas y validación

Tras el desarrollo de los módulos, se realizaron varias pruebas para asegurar la estabilidad, exactitud y eficiencia del sistema. La validación se realizó tanto de forma técnica como operativa, en un esquema de mejora continua impulsado por la retroalimentación del usuario final.

Durante el desarrollo, cada función se probó de forma aislada dentro de cuadernos Jupyter, comparando su comportamiento con los resultados históricos del proceso manual. Esta estrategia permitió identificar inconsistencias de manera temprana, asegurando que las nuevas automatizaciones replicaran correctamente la lógica de negocio ya establecida.

De manera paralela, se entregaron versiones funcionales parciales al colaborador responsable del proceso, quien evaluó el sistema en situaciones reales. Esta verificación temprana

facilitó la detección de errores contextuales y la mejora de la experiencia de usuario.

Las pruebas realizadas incluyeron:

- **Pruebas funcionales:** Verificación de que cada módulo cumpliera con los requisitos definidos (extracción, consulta externa, aplicación de políticas, generación de reportes).
- **Pruebas de validación de datos:** Comparación de los resultados generados automáticamente frente a los obtenidos manualmente en procesos anteriores.
- **Pruebas de rendimiento:** Evaluación de tiempos de ejecución de los subprocesos, eficiencia del *multithreading* y estabilidad general del sistema ante múltiples solicitudes.
- **Pruebas de usuario:** Revisión directa por parte del colaborador, evaluando la usabilidad de la interfase y la lógica del flujo de trabajo.

Como resultado, se ajustaron elementos de diseño visual, se mejoró la organización de las salidas y se afinaron reglas internas de validación, hasta consolidar una versión estable y alineada con las necesidades reales del equipo de cobranza.

E. Fase 5: Despliegue y capacitación

Una vez finalizadas las etapas de desarrollo y validación, se procedió al despliegue de la herramienta en el entorno real de trabajo del equipo de prejurídicos del banco. La instalación se realizó en los equipos de los colaboradores directamente involucrados en el proceso de investigación de bienes.

Se realizaron sesiones de capacitación personalizadas con el usuario clave, enfocadas en:

- La operación general del aplicativo.
- La lógica del flujo semi-automatizado.
- La correcta interpretación de los mensajes generados en consola.

Como parte del proceso de entrega, se desarrolló una **documentación interna de usuario**, elaborada en formato PDF, que incluye instrucciones paso a paso sobre el uso de cada módulo, soluciones a errores frecuentes y recomendaciones para mantener la integridad del sistema. Este material fue diseñado para facilitar la adopción por parte de otros miembros del equipo y asegurar la continuidad del uso de la herramienta más allá del periodo de esta práctica.

Adicionalmente, se establecieron rutas compartidas para la generación y almacenamiento de resultados, garantizando su trazabilidad y disponibilidad para todos los actores involucrados.

Gracias a este despliegue controlado y documentado, el equipo de cobranza pudo comenzar a operar el sistema con confianza, reduciendo la curva de aprendizaje y asegurando una transición efectiva desde el proceso manual hacia una operación automatizada.

VII. RESULTADOS

La implementación del aplicativo **"Investigación de Bienes, Estructuración y Preparación"** ha representado un avance significativo en la automatización del proceso de investigación de bienes en el área de cobranza del banco. A continuación, se presentan los principales resultados obtenidos tras el despliegue de la herramienta:

- Desde el área de **Torre de Control**, se gestionan diariamente entre **100 y 1000 nuevos clientes** que requieren estructuración jurídica. El aplicativo ha permitido agilizar este proceso, integrando múltiples tareas en una única interfase centralizada.
- El equipo interno de prejurídico genera entre **20 y 100 solicitudes diarias** de investigación o actualización de documentos, como certificados de tradición o matrículas mercantiles. Gracias al aplicativo, estas solicitudes se pueden procesar de forma automática, disminuyendo la carga operativa del equipo y mejorando los tiempos de respuesta.
- La herramienta ha permitido iniciar una **migración gradual del proceso tercerizado hacia una operación interna**, disminuyendo la dependencia del proveedor externo encargado de las investigaciones. Aunque esta transición aún está en curso, se ha empezado a procesar una parte significativa de las solicitudes directamente con el aplicativo, validando su funcionamiento en escenarios reales.
- El sistema se encuentra actualmente en estado de **"EN DESARROLLO"**, lo cual permite su mejora continua bajo un enfoque iterativo. Esto incluye la recolección de retroalimentación directa por parte de los usuarios, el ajuste de módulos existentes y la incorporación progresiva de nuevas funcionalidades.
- La trazabilidad de cada solicitud y su resultado ha mejorado significativamente, gracias al almacenamiento estructurado en bases de datos SQLite y a la separación de tareas en la interfase gráfica. Esto ha permitido una mejor gestión documental, más claridad

sobre el estado de cada proceso y mayor control sobre el cumplimiento de las políticas de decisión definidas por el banco.

Estos resultados reflejan un impacto directo tanto en la eficiencia operativa del proceso como en la autonomía tecnológica del área, habilitando nuevas capacidades internas que antes estaban restringidas por la dependencia de aliados externos.

En términos de tiempo, las ganancias que experimenta el área medidas en minutos para sus distintos procesos se presentan en la figura 4.

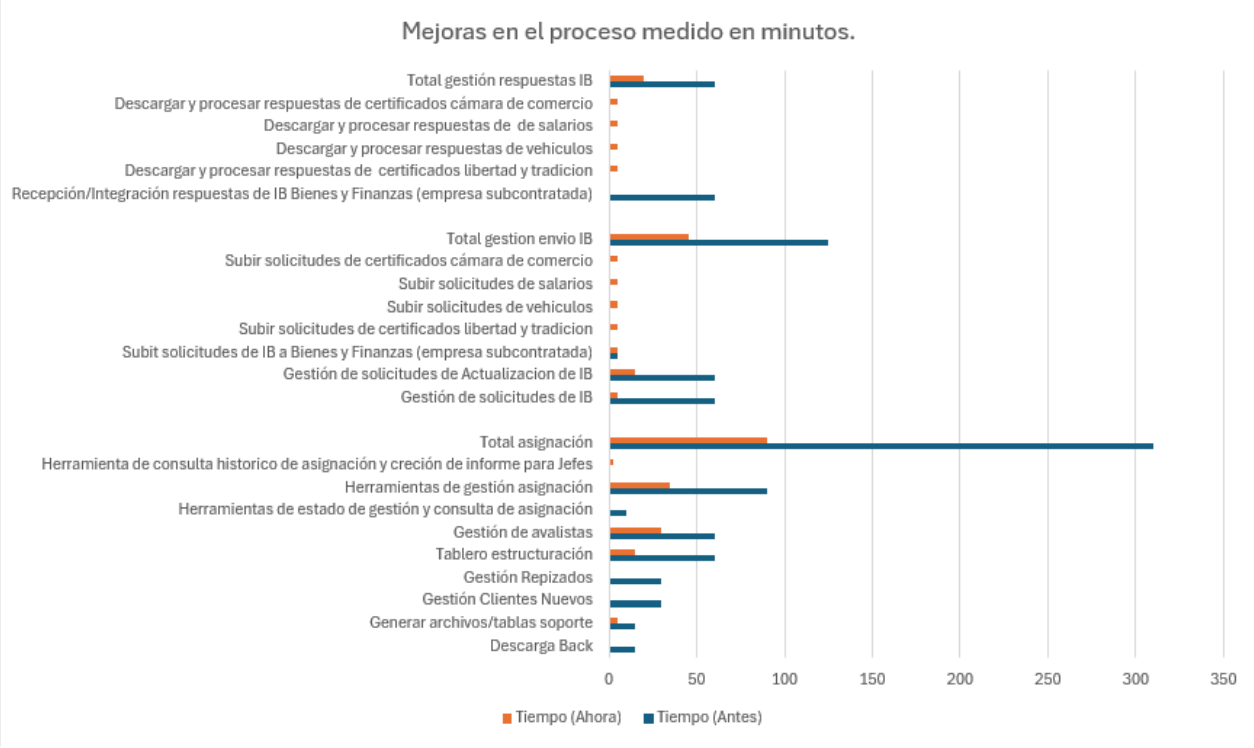


Fig. 4. Impactos sobre el proceso de IB EyP.

VIII. CONCLUSIONES

El desarrollo e implementación de la herramienta **"Investigación de Bienes, Estructuración y Preparación"** representó una respuesta efectiva a una necesidad concreta del área de cobranza del banco: reducir la dependencia de terceros y mejorar el control interno sobre el proceso de investigación de bienes y preparación de demandas judiciales.

Entre las principales conclusiones se destacan:

- La automatización del proceso permitió reducir significativamente la carga operativa del equipo, eliminando tareas repetitivas que antes se realizaban manualmente mediante hojas de cálculo y bases de datos dispersas.
- El uso de técnicas de *web scraping*, la implementación de una arquitectura modular en Python y la incorporación de *multithreading* permitieron construir una herramienta robusta, flexible y con una experiencia de usuario fluida.
- La validación continua mediante cuadernos Jupyter y la entrega iterativa de versiones de prueba facilitaron la detección temprana de errores y el alineamiento funcional con las necesidades reales del usuario final.
- La migración gradual del proceso hacia una operación interna ya está en marcha, y aunque el sistema aún se encuentra registrado como **"EN DESARROLLO"**, ha demostrado su utilidad y confiabilidad en escenarios reales, sirviendo como eje para futuras mejoras y escalabilidad.
- Esta experiencia demuestra el valor que pueden generar los procesos de transformación digital liderados desde dentro de la organización, promoviendo la autonomía tecnológica, el ahorro de recursos y la mejora en la toma de decisiones.
- Desde el punto de vista académico y profesional, el proyecto permitió aplicar conocimientos en programación, automatización, bases de datos y diseño de interfaces, al tiempo que fortaleció habilidades de análisis de procesos, trabajo colaborativo y entrega de valor en un entorno corporativo real.

Bibliografía

[1] S. F. de Colombia, "Guía de buenas prácticas para la gestión de la cartera," https://www.superfinanciera.gov.co, 2019, accessed: 2025-05-14.

[2] B. de la República, "Manual de cobranza de cartera en entidades financieras colombianas," https://www.banrep.gov.co, 2020, accessed: 2025-05-14.

[3] R. Mitchell, *Web Scraping with Python: Collecting Data from the Modern Web*, 2nd ed. Sebastopol, CA: O'Reilly Media, 2018.

[4] L. Richardson, "Beautiful Soup Documentation," https://www.crummy.com/software/BeautifulSoup/, 2023, accessed: 2025-05-14.

[5] SeleniumHQ, "Selenium WebDriver Documentation," https://www.selenium.dev/documentation/, 2023, accessed: 2025-05-14.

[6] R. T. Fielding, *Architectural Styles and the Design of Network-based Software Architectures (Tesis doctoral sobre REST APIs)*, 2000.

[7] "2024 state of the API report," https://www.postman.com/state-of-api/, accessed: 2025-6-3.

[8] M. Owens, *The Definitive Guide to SQLite*. New York, NY: Apress, 2010.

[9] B. Ltd. (2024) Bizagi modeler 4.2 service overview. Consultado el 4 de junio de 2025. [Online]. Available: https://help.bizagi.com/platform/es/intro_welcome.htm

[10] L. Learning. (2025) Método moscow: Prioriza funcionalidades y proyectos. Consultado el 5 de junio de 2025. [Online]. Available: https://lemonlearning.com/es/blog/metodo-moscow

[11] Wikipedia, "MoSCoW method," 2024, consultado el 5 de junio de 2025. [Online]. Available: https://en.wikipedia.org/wiki/MoSCoW_method

[12] E. Miranda, "Moscow Rules: A quantitative exposé," *PM World Journal*, vol. XII, no. III, 2023, carnegie Mellon University. [Online]. Available: https://pmworldjournal.com/wp-content/uploads/2023/03/pmwj127-Mar2023-Miranda-Moscow-Rules-a-quantitative-expose-2nd-ed.pdf

[13] S. Vijayakumar, K. P. K., and R. H. M., "Assessing the Effectiveness of MoSCoW Prioritization in Software Development: A Holistic Analysis across Methodologies," *EAI Endorsed Transactions on Internet of Things*, vol. 10, 2024, doi:10.4108/eetiot.6515. [Online]. Available: https://eudl.eu/doi/10.4108/eetiot.6515

[14] J. E. Grayson, *Python and Tkinter Programming*. Manning Publications, 2000.

[15] TkDocs, "Tkinter Tutorial," https://tkdocs.com/tutorial/, 2023, accessed: 2025-05-14.

[16] M. Dumas, M. La Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management*, 2nd ed. Springer, 2018.

[17] I. C. Education, "What is Process Orchestration?" https://www.ibm.com/cloud/learn/process-orchestration, 2021, accessed: 2025-05-14.

[18] Deloitte, "Digital Banking Maturity Report," https://www2.deloitte.com, 2021, accessed: 2025-05-14.

[19] B. I. de Desarrollo, "Transformación digital de la banca en América Latina," https://publications.iadb.org, 2020, accessed: 2025-05-14.

[20] A. Chapagain, *Hands-On Web Scraping with Python: Perform advanced scraping operations using various Python libraries and tools such as Selenium, Regex, and others.* Packt Publishing Ltd, 2019.

[21] R. Gunawan, A. Rahmatulloh, I. Darmawan, and F. Firdaus, "Comparison of web scraping techniques: regular expression, HTML DOM and Xpath," *International Conference on Industrial Enterprise and System Engineering*, vol. 2, pp. 283–287, 2018.