Ciencia Latina Revista Científica Multidisciplinar, Ciudad de México, México.
ISSN 2707-2207 / ISSN 2707-2215 (en línea), julio-agosto 2026,
Volumen 10, Número 4.

https://doi.org/10.37811/cl_rcm.v10i4

APLICACIÓN DE ESCRITORIO PARA LA GESTIÓN DE PLANES Y HORARIOS ACADÉMICOS EN LA COORDINACIÓN DE POSGRADOS EN LA MAESTRÍA EN INGENIERÍA PARA LA INNOVACIÓN Y DESARROLLO TECNOLÓGICO

DESKTOP APPLICATION FOR THE MANAGEMENT OF ACADEMIC PLANS AND SCHEDULES IN GRADUATE STUDIES COORDINATION IN THE MASTER'S DEGREE IN ENGINEERING FOR INNOVATION AND TECHNOLOGICAL DEVELOPMENT

Marcos David Chino Cuamacateco
Universidad Autónoma de Guerrero

Lucia Flores Ibañez
Universidad Autónoma de Guerrero

Ing. Cristian Omar Torres Chegue
Universidad Autónoma de Guerrero

Dr. Angelino Feliciano Morales
Universidad Autónoma de Guerrero

Dr. Rene Edmundo Cuevas Valencia
Universidad Autónoma de Guerrero

DOI: https://doi.org/10.37811/cl_rcm.v10i4.25274

Aplicación de escritorio para la gestión de planes y horarios académicos en la coordinación de posgrados en la Maestría en Ingeniería para la Innovación y Desarrollo Tecnológico

Marcos David Chino Cuamacateco[1]
chinocuamacatecomarcosdavid@gmail.com
https://orcid.org/0009-0007-1637-3900
Universidad Autónoma de Guerrero
México

Lucia Flores Ibañez
luciafloresibanez234@gmail.com
https://orcid.org/0009-0006-9623-0450
Universidad Autónoma de Guerrero
México

Ing. Cristian Omar Torres Chegue
13005354@uagro.mx
https://orcid.org/0009-0000-1798-0912
Universidad Autónoma de Guerrero
México

Dr. Angelino Feliciano Morales
afmorales@uagro.mx
http://orcid.org/0000-0002-7707-7319
Universidad Autónoma de Guerrero
México

Dr. Rene Edmundo Cuevas Valencia
reneecuevas@uagro.mx
http://orcid.org/0000-0001-9528-7603
Universidad Autónoma de Guerrero
México

RESUMEN

La administración de horarios en posgrados de ingeniería implica resolver simultáneamente restricciones de disponibilidad docente, asignación de aulas, distribución semestral de materias, distinción entre tronco común y optativas, y coordinación entre Líneas de Investigación e Incidencia Social (LIES). Este artículo describe la implementación de una aplicación de escritorio en Python con Flet, que automatiza la gestión de planes de estudio y horarios en la Coordinación de Posgrado de la Maestría en Ingeniería para la Innovación y Desarrollo Tecnológico (MIIDT) de la Universidad Autónoma de Guerrero. El sistema adopta una arquitectura en capas y aplica Domain-Driven Design (DDD): entidades con reglas explícitas, objetos de valor inmutables, especificaciones componibles y reglas de validación encapsuladas. La persistencia se realiza con SQLAlchemy en MySQL, y la exportación de horarios con ReportLab en PDF. La metodología siguió un enfoque iterativo-incremental, con un backlog de ocho épicas ejecutado en seis sprints. Se espera que el sistema reduzca un 90 % el tiempo requerido para la elaboración de horarios y elimine los conflictos de traslape mediante validación automática en tiempo real. La validación técnica del software se realizó mediante 77 pruebas automatizadas desarrolladas con Pytest, incluyendo pruebas unitarias, de integración y de fuerza bruta, todas ejecutadas exitosamente.

Palabras clave: aplicación; gestión; administración; MIIDT; DDD.

[1]Autor principal.
Correspondencia: reneecuevas@uagro.mx

Desktop Application for the Management of Academic Plans and Schedules in Graduate Studies Coordination in the Master's Degree in Engineering for Innovation and Technological Development

ABSTRACT

Managing schedules in engineering graduate programs requires simultaneously solving constraints of faculty availability, classroom allocation, semester-based course distribution, distinction between core and elective courses, and coordination among Research and Social Impact Lines (LIES). This article describes the implementation of a desktop application built in Python with Flet, which automates the management of study plans and schedules at the Graduate Studies Coordination of the MIIDT at the Universidad Autónoma de Guerrero. The system adopts a layered architecture and applies Domain-Driven Design (DDD): entities with explicit invariants, immutable value objects, composable specifications, and encapsulated validation rules. Persistence is handled with SQLAlchemy over MySQL, and schedule exports are produced with ReportLab in PDF. The methodology followed an iterative-incremental approach, with a backlog of eight epics executed across six sprints. The system is expected to reduce the time required for schedule preparation by 90% and eliminate overlap conflicts through real-time automatic validation. The software's technical validation was carried out through 77 automated tests developed with Pytest, including unit, integration, and brute-force tests, all executed successfully.

Keywords: implementation; management; administration; MIIDT; DDD.

Artículo recibido 13 mayo 2026
Aceptado para publicación: 21 junio 2026

INTRODUCCIÓN

La programación de horarios en las universidades es, sin duda, una de las tareas administrativas más complicadas de resolver: se trata esencialmente de un problema de satisfacción de restricciones (Constraint Satisfaction Problem, CSP), definido por Russell y Norvig (2020) como aquel en el que un conjunto de variables debe tomar valores dentro de un dominio finito sujetos a restricciones que limitan las combinaciones válidas entre ellas, ya que exige coordinar al mismo tiempo distintos recursos (docentes, aulas, periodos escolares y planes de estudio) cada uno sujeto a sus propias condiciones de disponibilidad, compatibilidad y orden de precedencia. Según Bashab et al. (2023), este problema se vuelve difícil en el nivel posgrado, donde influyen factores como la variedad de líneas de especialización, las cargas de trabajo distintas entre docentes de tiempo completo y de asignatura, y la necesidad de encajar las materias obligatorias del tronco común junto con las optativas.

La coordinación de posgrado de la Maestría en Ingeniería para la Innovación y Desarrollo Tecnológico (MIIDT) de la Universidad Autónoma de Guerrero (UAGro) desde su creación en 2014 a operado mediante procedimientos manuales en hojas de cálculo sin validación automática de restricciones (traslape de horarios, asignación de aulas a varios docentes al mismo tiempo ,etc.), lo que generaba errores frecuentes de traslape, inconsistencias entre planes de estudio y horarios publicados, y un alto costo en tiempo administrativo.

Este trabajo describe la implementación de una solución que automatiza el ciclo completo: creación de planes de estudio con sus líneas de especialización, asignación de materias a semestres, registro de horarios con validación automática de restricciones en tiempo real, historial de horarios por periodo escolar, consulta de carga horaria por docente y exportación de documentos PDF institucionales. Después de analizar las diferentes propuestas de solución se llegó a la conclusión que la opción más adecuada es una aplicación de escritorio para Windows construida con Python y Flet (motor de renderizado de Flutter), con persistencia en MySQL a través de SQLAlchemy y una arquitectura en capas cuyos principios de Domain-Driven Design siguen Özkan et al. (2025).

Antecedentes. Los sistemas de gestión de horarios universitarios se remontan a los años 1960, cuando los primeros algoritmos de backtracking se aplicaron a la planificación de exámenes (citados en Bashab et al., 2023). En las décadas de 1980 y 1990 surgieron los primeros Sistemas de Información de

Estudiantes (SIS). Akour y Alenezi (2022) documentan la tendencia más amplia de transformación digital de la educación superior a nivel mundial.

En la primera década del siglo XXI emergieron plataformas de segunda generación: Banner (Ellucian), PeopleSoft Campus Solutions (Oracle) y SAP Student Lifecycle Management. Akour y Alenezi (2022) señalan que la adopción de este tipo de sistemas institucionales de gran escala en la educación superior está condicionada por su alto costo de licenciamiento y por la complejidad de su configuración, lo que las vuelve poco accesibles para coordinaciones de posgrado con recursos limitados, como es el caso de la MIIDT.

En el software libre, el proyecto FET (Free Timetabling Software) (Lalescu, 2024) resuelve el UCTP mediante algoritmos heurísticos de asignación automática, pero está orientado principalmente a educación básica y media, y carece de gestión de LIES, distinción tronco común/optativa, integración con planes de estudio formales y generación de PDF con membrete personalizable.

Useche et al. (2022) observan que la adopción de sistemas de programación especializados en América del Sur sigue siendo limitada en comparación con las plataformas generales, y algunos problemas en la especialización no están claros. Por ejemplo, en México, el Sistema Integral de Información del TecNM incluye horarios de licenciatura, pero no las particularidades de posgrado de SECIHTI, y en Guerrero el Sistema Integral de Administración Escolar (SIAE) de la UAGro tampoco incluye LIES, categorización de materias por tipo, o múltiples cohortes en el mismo plan de posgrado, lo cual es el tipo de brecha funcional que motivó este proyecto.

Desde la ingeniería de software, Sangabriel-Alarcón et al. (2024) establecen el Diseño Guiado por el Dominio (DDD): la lógica de negocio debe residir en el núcleo del dominio (en entidades con su propia identidad y reglas, objetos de valor inmutables y especificaciones claras).

Santiago-Salazar y Rico-Bautista (2024) complementan este enfoque con evidencia empírica sobre la Arquitectura Limpia: las capas están diseñadas de tal manera que sus dependencias siempre apuntan hacia adentro (hacia el dominio) para aumentar la mantenibilidad en comparación con arquitecturas sin esta separación. Este proyecto hace esto tal como se establece en el código fuente: los comentarios de prohibición al inicio de cada módulo de dominio *('PROHIBIDO: importar Flet aquí', 'PROHIBIDO: acceder a la base de datos aquí')* son la manifestación concreta de la inversión de dependencias.

Estado del arte. Bashab et al. (2023) revisaron 47 algoritmos de solución para el problema de asignación de horarios universitarios conocido como University Course Timetabling Problem (UCTP, por sus siglas en inglés) y concluyeron que los enfoques metaheurísticos (algoritmos de búsqueda inteligente que exploran muchas combinaciones posibles sin garantizar la mejor solución, pero que la encuentran en tiempos razonables), en el que se aborda los algoritmos genéticos, la búsqueda tabú (algoritmos de optimización) y las colonias de hormigas (técnicas inspiradas en la naturaleza), superan a los métodos exactos en instancias de tamaño real; el 62 % de los sistemas revisados aplican restricciones duras y blandas.

De estos hallazgos se retoma únicamente la clasificación de restricciones: el sistema aquí desarrollado valida solo restricciones duras, en tiempo real y mediante reglas de dominio explícitas; la incorporación de metaheurísticas para la generación automática de horarios queda fuera de alcance y se plantea como trabajo futuro.

Ceschia et al. (2023) realizan una revisión sistemática de las formulaciones estándar y los bancos de pruebas (benchmarks) del problema de asignación de horarios educativos, identificando seis formulaciones de referencia y reportando resultados del estado del arte en calidad de solución, técnicas de búsqueda y tiempos de ejecución, este trabajo se retoma como referencia de los formatos y restricciones típicas del problema de horarios que el sistema debe representar.

Zhong et al. (2024) encuentran que los sistemas con arquitectura en capas y patrón *Repository* son fáciles de mantener que los sistemas sin esta separación. Con base en ese hallazgo, la arquitectura implementada separa explícitamente las capas de Aplicación e Infraestructura, con el patrón *Repository* cubriendo todo el acceso a datos. Sangabriel-Alarcón et al. (2024), en su revisión sistemática, documentan que el Diseño Guiado por el Dominio (DDD) mejora la capacidad de mantenimiento cuando las reglas de negocio son complejas y evolutivas, hallazgo aplicable a coordinaciones con múltiples LIES.

Percival y Gregory (2020), en su obra de referencia sobre patrones arquitectónicos en Python, muestran que el patrón de Repositorio (*Repository*) junto con SQLAlchemy mantiene el dominio libre de dependencia de infraestructura, lo que facilita las pruebas al reemplazar la base de datos real con simulaciones en pruebas unitarias. Ese principio es el que permite, en la implementación descrita en el

apartado de la metodología, que las clases *HorarioRepository* y *PlanesRepository* encapsulen el acceso a MySQL sin que el dominio dependa de SQLAlchemy.

Useche et al. (2022) consideran que la llegada de la pandemia permitió la aceleración de la transformación digital en instituciones latinoamericanas. Bajo ese panorama es en el que se ubica la necesidad de la MIIDT de modernizar un proceso administrativo que seguía dependiendo de hojas de cálculo sin automatización. García Yáñez et al. (2024) informan sobre una plataforma de programación web para una universidad en Tampico, México, que reduce los errores de superposición. Romaguera et al. (2024) describen un sistema web basado en un algoritmo genético mejorado cuya separación entre la lógica de asignación y la interfaz facilita la iteración.

Ambos antecedentes comparten el objetivo de reducir errores de traslape mediante software, pero ninguno modela las particularidades de un posgrado con Líneas de Investigación e Incidencia Social (LIES) ni la distinción entre tronco común y optativas. Esa brecha funcional, no cubierta por ninguno de los dos, es la que se atiende de forma específica en el sistema desarrollado (la comparación se retoma con detalle más adelante).

Flet (2024) encuentra tiempos de renderizado de menos de 16 ms gracias al motor Flutter con un modelo de componentes reactivo que prefiere separar la presentación de la lógica de negocio. Por ese motivo se seleccionó Flet frente a Tkinter, PyQt5 o wxPython (otras bibliotecas para construir interfaces de escritorio en Python) para construir la capa de presentación. Wahyudi et al. (2022) destacaron las ventajas de MySQL para instituciones con recursos limitados, debido a que no requiere costos de licenciamiento, soporta grandes volúmenes de información y ofrece compatibilidad multiplataforma, características que justifican su selección en este proyecto.

Lukasczyk et al. (2023) demostraron que combinar técnicas de generación de pruebas en Python mejora la cobertura (el porcentaje del código que efectivamente se ejecuta y verifica durante las pruebas) y ayuda a detectar errores antes de la integración, lo que respalda la estrategia de pruebas unitarias, de integración y de fuerza bruta adoptada en el desarrollo.

Özkan et al. (2025) concluyeron que, para organizaciones medianas, los sistemas monolíticos (aplicaciones construidas como un solo programa unificado, más simples de instalar y mantener) con buena separación de capas y DDD son más prácticos que las arquitecturas de microservicios, en términos

de despliegue, costo e infraestructura. Sobre esa base se decidió construir el sistema como una aplicación monolítica de escritorio en lugar de una arquitectura distribuida o de microservicios, dado el tamaño de la coordinación y sus recursos institucionales.

Resumen del Estado del Arte. La Tabla 1 resume los antecedentes y trabajos del estado del arte revisados, y para cada uno el autor, el nombre del trabajo, una breve descripción de cómo se realizó este trabajo, las herramientas utilizadas y la relevancia del trabajo dentro del campo.

Tabla 1. Resumen de antecedentes y estado del arte.

Autor	Nombre del trabajo	Descripción del trabajo	Herramientas utilizadas	Vigencia
Akour y Alenezi (2022)	Higher education future in the era of digital transformation	Estudian la tendencia global de transformación digital en la educación superior y el costo/complejidad de los SIS de gran escala	Revisión documental	2022 – vigente, referencia general de contexto
Bashab et al. (2023)	Optimization techniques in university timetabling problem	Revisión de 47 algoritmos metaheurísticos para el UCTP; clasifica restricciones duras y blandas	Revisión sistemática de literatura	2023 – vigente, área de investigación activa
Ceschia et al. (2023)	Educational timetabling: Problems, benchmarks, and state-of-the-art results	Revisión de las formulaciones actuales y evaluación comparativa del problema de programación educativa en su conjunto.	Revisión sistemática de literatura	2023 – vigente
Flet (2024)	Flet: Build multi-platform apps in Python powered by Flutter	Documentación oficial del framework de interfaz gráfica utilizado para la capa de presentación	Flet / Flutter (Python)	2024 – vigente, versión activa (0.28)
García Yáñez et al. (2024)	Development of a web-based timetabling software	Plataforma de programación web para una universidad mexicana que reduce errores de superposición	Desarrollo web (working paper)	2024 – vigente

Autor	Nombre del trabajo	Descripción del trabajo	Herramientas utilizadas	Vigencia
	for a Mexican university			
Lukasczyk et al. (2023)	An empirical study of automated unit test generation for Python	Estudio empírico sobre la generación automatizada de pruebas unitarias en Python y su impacto en la cobertura	Herramientas de generación de pruebas en Python	2023 – vigente
Özkan et al. (2025)	Domain-Driven Design in software development: A systematic literature review	Revisión sistemática sobre implementación, desafíos y efectividad del DDD, incluyendo su aplicabilidad a sistemas monolíticos	Revisión sistemática de literatura	2025 – vigente, la más reciente del corpus
Percival y Gregory (2020)	Architecture patterns with Python	Trabajo de referencia sobre patrones arquitectónicos (Repositorio, Capa de servicio) en Python con TDD y DDD	Python, SQLAlchemy	2020 – vigente como referencia arquitectónica
Romaguera et al. (2024)	Development of a web-based course timetabling system based on an enhanced genetic algorithm	Sistema de programación web basado en un algoritmo genético mejorado, con separación entre lógica de asignación e interfaz	Algoritmo genético, desarrollo web	2024 – vigente
Sangabriel-Alarcón et al. (2024)	Domain-Driven Design in microservices-based systems development	Revisión sistemática y análisis temático sobre DDD y su efecto en la mantenibilidad de reglas de negocio complejas	Revisión sistemática de literatura	2024 – vigente
Santiago-Salazar y Rico-Bautista (2024)	Clean Architecture: Impact on Performance and Maintainability of	Evidencia empírica sobre Arquitectura Limpia y el cambio de dependencias hacia el dominio	Estudio empírico de caso	2024 – vigente

Autor	Nombre del trabajo	Descripción del trabajo	Herramientas utilizadas	Vigencia
	Native Android Projects			
Useche et al. (2022)	Reflexive pedagogy at the heart of educational digital transformation in Latin American higher education institutions	Examinan la aceleración de la transformación digital educativa en Latinoamérica y la adopción limitada de software especializado	Revisión documental	2022 – vigente
Wahyudi et al. (2022)	Database Management Education in MYSQL	Muestran las ventajas de MySQL (sin costo de licencia, escalabilidad, multiplataforma) con requisitos mínimos y un bajo costo para pequeñas organizaciones	MySQL	2022 – vigente
Zhong et al. (2024)	Domain-Driven Design for microservices: An evidence-based investigation	Evidencia que las arquitecturas en capas con el patrón Repository (Repositorio) son más fáciles de mantener	Investigación basada en evidencia	2024 – vigente

Después de la revisión y considerados en conjunto, estos antecedentes y trabajos del estado del arte determinaron de manera directa las decisiones de diseño del sistema desarrollado. La revisión de plataformas comerciales de gran escala y del proyecto FET (Antecedentes) evidenció una brecha funcional específica de posgrado que ninguna solución existente, incluidas las de García Yáñez et al. (2024) y Romaguera et al. (2024), cubre por completo, lo que justificó construir una aplicación propia orientada a la MIIDT. De Bashab et al. (2023) y Ceschia et al. (2023) se retomó la clasificación de restricciones duras y las formulaciones típicas del problema de horarios, sin adoptar metaheurísticas de generación automática.

De Zhong et al. (2024), Sangabriel-Alarcón et al. (2024) y Santiago-Salazar y Rico-Bautista (2024) se tomaron los principios de arquitectura en capas, DDD y Arquitectura Limpia que estructuran el sistema (Dominio, Aplicación, Infraestructura, Presentación). Percival y Gregory (2020) fundamentaron el uso del patrón *Repository* sobre SQLAlchemy, mientras que Flet (2024) y Wahyudi et al. (2022) sustentaron la selección tecnológica de la capa de presentación y de la base de datos, respectivamente. Finalmente, Lukasczyk et al. (2023) y Özkan et al. (2025) orientaron, en conjunto, la estrategia de pruebas automatizadas y la decisión de construir una aplicación monolítica de escritorio en vez de una arquitectura distribuida.

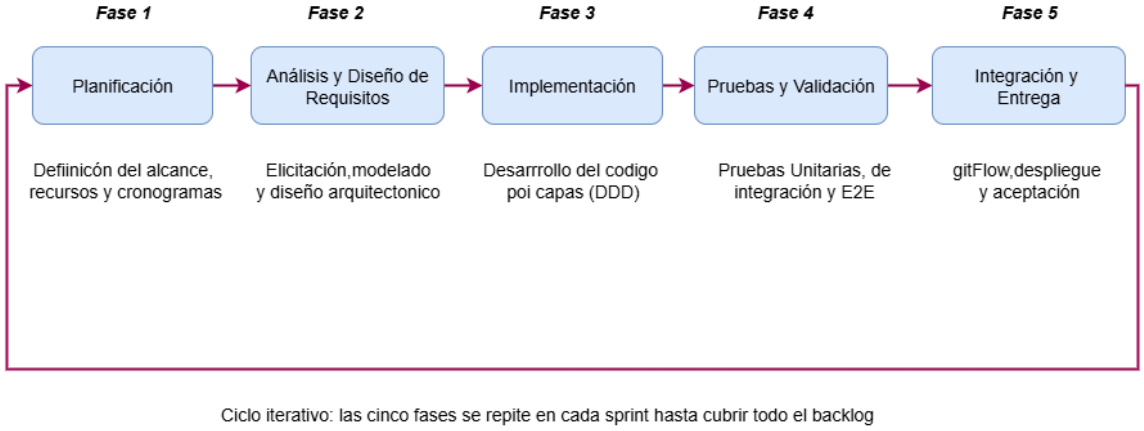
Problemática. La problemática de la coordinación de posgrado en la MIIDT incluía cinco dimensiones: (1) construcción manual de horarios en hojas de cálculo sin validación automática, con traslapes frecuentes de docentes y aulas; (2) desvinculación entre el plan de estudios formal y los horarios operativos, construidos de forma independiente; (3) ausencia de historial estructurado de horarios por periodo escolar; (4) tiempo de elaboración de tres a cinco días hábiles por periodo; y (5) generación manual de documentos PDF de difusión sin membrete institucional estandarizado.

Objetivos. El objetivo principal de este trabajo es crear una aplicación de escritorio para la gestión de planes de estudio y horarios académicos en la Coordinación de Posgrado de la Maestría en Ingeniería para la Innovación y el Desarrollo Tecnológico (MIIDT) en la Universidad Autónoma de Guerrero (UAGro) para la validación en tiempo real de restricciones y la exportación de documentos estandarizados. Para lograr esto, proponemos modelar el sistema académico definiendo los elementos, relaciones y reglas de negocio; construir una estructura en capas que favorezca la organización del software; desarrollar un mecanismo de validación que encuentre automáticamente superposiciones entre las materias de tronco común y optativas considerando diferentes Líneas de Investigación e Incidencia Social (LIES); integrar un módulo para la generación de documentos en formato PDF con diseño institucional; y validar el funcionamiento del sistema utilizando pruebas de software para asegurar la calidad y fiabilidad de la aplicación.

METODOLOGÍA

El proyecto se desarrolló bajo un enfoque iterativo e incremental cuyas cinco fases (Planificación; Análisis y Diseño de Requisitos; Implementación; Pruebas y Validación; e Integración y Entrega) se ejecutaron de forma cíclica en cada sprint (Figura 1).

Figura 1: Fases generales de la metodología iterativa e incremental aplicada al proyecto. Fuente de elaboración propia



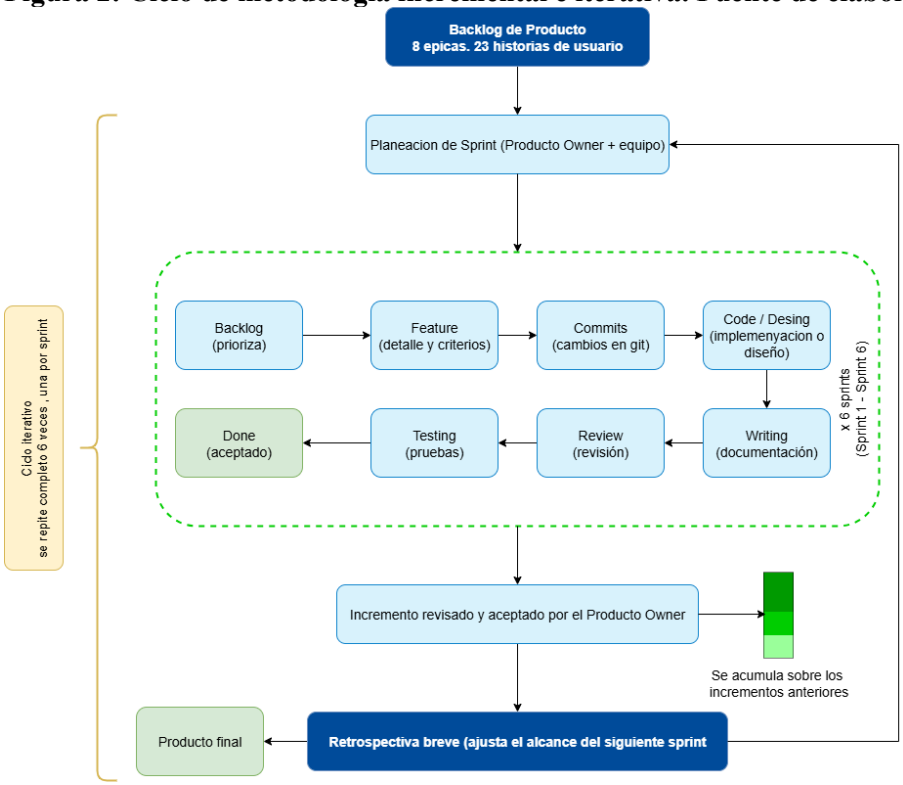
A continuación, se describen las fases generales de la metodología iterativa-incremental aplicada al proyecto y mencionando la administración bajo la metodología scrum.

Fase 1: Planificación. El desarrollo se llevó a cabo mediante trabajo cercano y continuo con la Coordinación de Posgrado de la MIIDT durante aproximadamente tres meses, en bloques de trabajo cuyo incremento era revisado y priorizado por la coordinación al cierre de cada uno. En esta fase se definió el alcance, se identificaron las líneas de trabajo (planes de estudio, catálogos, motor de validación, generación de PDF) y se organizó el trabajo en épicas e historias de usuario. Todo el desarrollo se realizó en Python, Liu et al. (2024) describen esta adopción generalizada y la gran cantidad de bibliotecas y pruebas (incluyendo Pytest) disponible en Python 3.13; también proporcionan soporte completo de types hints (sugerencias de tipo), que aparecen en las firmas de los métodos públicos en el sistema.

Enfoque metodológico global. Lima et al. (2023) describen la filosofía general de desarrollo como iterativa e incremental: cada sprint producía software funcional, probado y posiblemente entregable. Esta filosofía fue implementada por Scrum (Schwaber y Sutherland, 2020): un backlog de ocho épicas en seis sprint de 10 días desde el 9 de marzo hasta el 5 de junio de 2026 en un tablero Kanban de ocho columnas en Trello *(Backlog, Feature, Commits, Code/Design, Writing, Review, Testing, Done)*, ilustrado en la Figura 2. Un miembro asumió el rol de Product Owner y el resto del equipo compartió

los roles de Scrum Master y equipo de desarrollo, con ceremonias ligeras: planificación al inicio de un sprint, seguimiento continuo en el tablero, revisión con el Product Owner al final de cada bloque, y algunas retrospectivas.

Figura 2: Ciclo de metodología incremental e iterativa. Fuente de elaboración propia



El backlog agrupó 23 historias de usuario (HU1–HU23) en ocho épicas (Tabla 2), derivadas de los módulos funcionales identificados en la recolección de requisitos.

Tabla 2. Épicas del backlog de producto (8 épicas, 23 historias de usuario).

Épica	Nombre	Alcance	Historias de usuario
EC1	Definición Técnica	Selección justificada del stack tecnológico y configuración del entorno de desarrollo compartido	HU1, HU2
EC2	Diseño y UX	Diseño de las cuatro pantallas principales: Planes de Estudio, Crear Plan, Horario Docente e Historial	HU3–HU6
EC3	Base de Datos	Diagramas ER/DR, script SQL, pruebas de integridad en local y migración al servidor	HU7–HU10
EC4	Gestión de Planes	Creación, consulta, edición y validación de horarios	HU11–HU13, HU16

Épica	Nombre	Alcance	Historias de usuario
EC5	Documentación	Manual de usuario por pantalla y documentación técnica	HU14, HU15
EC6	Gestión de Horarios	Generación, visualización y exportación del horario por docente	HU17, HU18
EC7	Historial de Horarios	Consulta con filtros, gestión y exportación del historial	HU19–HU21
EC8	Pruebas y Calidad	Validación funcional del sistema en producción y corrección de errores	HU22, HU23

Fuente de elaboración propia.

La Tabla 3 detalla la distribución por sprint. El repositorio Git registra 20 ramas de tarea (task) en el mismo periodo, cada una anidada dentro de una rama sprint; la diferencia frente a las 23 historias es esperable, ya que algunas se resolvieron dentro de una misma rama.

Tabla 3. Distribución real del backlog en seis sprint de 10 días hábiles.

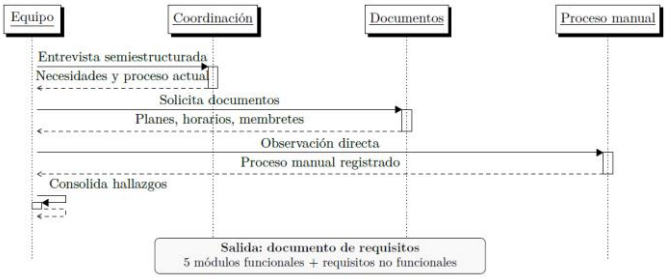
Sprint	Fechas	Objetivo principal
Sprint 1	9–20 mar. 2026	Stack tecnológico, diseño de las primeras pantallas y modelo de base de datos
Sprint 2	23 mar.–3 abr. 2026	Entorno de desarrollo, diseño restante, migración de BD y primer módulo funcional
Sprint 3	6–17 abr. 2026	Gestión de planes y horarios; arranque de la documentación
Sprint 4	20 abr.–2 may. 2026	Edición de planes, horario docente y cierre del manual de usuario
Sprint 5	4–22 may. 2026	Exportaciones y módulo completo de historial de horarios
Sprint 6	25 may.–5 jun. 2026	Validación funcional en producción y corrección de errores

Fuente de elaboración propia

Pressman y Maxim (2021) consideran este enfoque especialmente coherente en contextos académicos, donde los requisitos evolucionan conforme los usuarios validan incrementos funcionales en sesiones presenciales.

Fase 2: Análisis y Diseño de Requisitos. La recolección de requisitos empleó entrevistas semiestructuradas con la coordinación, análisis de documentos (planes de estudio vigentes, horarios históricos de cuatro periodos y formatos de membrete) y observación directa del proceso manual. Los requisitos funcionales se agruparon en cinco módulos: (1) gestión de planes de estudio con niveles, semestres, LIES y asignaciones materia-semestre; (2) gestión de recursos (docentes, aulas, periodos); (3) generación y validación de horarios con detección automática de conflictos; (4) historial de planes por periodo; y (5) exportación de PDF con timeline visual, tabla resumen y membrete personalizable. Los requisitos no funcionales incluyeron un tiempo de respuesta de menos de dos segundos, compatibilidad con Windows, instalación de aplicación sin servidor y código comprobable con cobertura automatizada.

Figura 3: Diagrama de secuencia de la recolección de requisitos en la Fase 2. Fuente de elaboración propia.



El diagrama (Figura 3) muestra cuatro actores (Equipo, Coordinación, Documentos y Proceso manual) con la siguiente secuencia, transcrita tal como aparece en el diagrama fuente: "Entrevista semiestructurada: Necesidades y proceso actual"; "Solicita documentos: Planes, horarios, membretes"; "Observación directa: Proceso manual registrado"; "Consolida hallazgos, Salida: documento de requisitos, 5 módulos funcionales + requisitos no funcionales".

Fase 3: Implementación – Arquitectura en Capas con DDD. Santiago-Salazar y Rico-Bautista (2024) y Sangabriel-Alarcón et al. (2024) describen los principios de una arquitectura en capas con flujo de dependencia unidireccional que se utilizan en el sistema (UI → Aplicación → Dominio ←

Infraestructura): la capa de Infraestructura implementa las interfaces definidas por Aplicación. (Tabla 4).

Tabla 4. Distribución de responsabilidades por capa arquitectónica.

Capa	Directorio raíz	Responsabilidades principales
Presentación	/ui	Vistas Flet, componentes reutilizables, navegador, builders de filas, generadores de PDF
Aplicación	/application	Servicios, casos de uso, DTOs, interfaces de repositorio, mapeadores, presentadores, contenedor DI
Dominio	/domain	Entidades, objetos de valor, especificaciones, reglas de validación, excepciones. Sin dependencias externas.
Infraestructura	/infrastructure	Repositorios SQLAlchemy, modelos ORM (15 tablas), conexión MySQL con python-dotenv

Fuente de elaboración propia.

Una regla explícita, documentada en los comentarios de cada módulo de dominio, prohíbe que cualquier archivo de Dominio importe Flet o acceda directamente a la base de datos; se verificó en cada revisión de código durante los seis sprint.

Implementación del Núcleo de Dominio: Constructos DDD. • Entidades. Özkan et al. (2025) señalan que, en DDD, las entidades deben encapsular sus propias invariantes en el constructor en lugar de delegar la validación a capas externas. Siguiendo ese principio, la entidad central es *Horario*, con *id_horario* como identidad (nulo hasta persistir). Sus reglas, validadas en el constructor, garantizan *hora_inicio* < *hora_fin* e *id_asignacion* > *0*. Implementa traslape(otro), *mismo_rango_que*(otro), *calcular_duracion()* y *es_tronco_comun()*. Docente y Aula son entidades secundarias representadas como DTOs (Objectos de transferencia de datos) en la capa de aplicación.

• Objetos de Valor. Conforme al principio de objetos de valor inmutables descrito por Sangabriel-Alarcón et al. (2024), estos elementos del sistema no tienen identidad propia. *DiaSemana* encapsula los días válidos mediante un tipo enumerado; Hora representa *HH:MM (24 h)* con validación de rango y

comparación completa, con *en_minutos()* para comparaciones de eficiencia. Periodo, Salón y Grupo siguen el mismo patrón.

• Especificaciones. Sangabriel-Alarcón et al. (2024) describen el patrón de Especificación con una clase base (Especificación) y las clases compuestas AndSpecification, OrSpecification y NotSpecification. Las especificaciones concretas son *HorarioDisponibleSpecification* (sin superposiciones) y *TroncoConsistenteEntreLiesSpecification* (asignaturas de tronco común con el mismo día, rango, aula y profesor entre LIES).

• Reglas de Validación (*HorarioValidator*). Siguiendo el hallazgo de Zhong et al. (2024) sobre la facilidad de modificación en las reglas de negocio cuando se encapsulan explícitamente fuera de las capas de presentación e infraestructura, el *HorarioValidator* concentra cuatro reglas de negocio con un enfoque equivalente al patrón Strategy: (1) *cross-LIES* de tronco, que exige coincidencia exacta en día, hora, aula y docente; (2) tronco versus tronco, sin traslape en el mismo día y semestre; (3) optativa versus tronco; y (4) optativa versus optativa dentro de la misma LIES. Toda divergencia lanza *HorarioConflictException*. Las validaciones operan en memoria con una latencia inferior a 500 ms por operación.

Implementación de la Capa de Aplicación. • Casos de Uso. De acuerdo con el patrón Service Layer descrito por Percival y Gregory (2020), los casos de uso encapsulan cada operación: *CrearHorarioUseCase*, *EditarHorarioUseCase*, *EliminarHorarioUseCase* y *GuardarPlanUseCase*. El flujo de creación construye la entidad Horario desde el DTO, consulta los horarios conflictivos, aplica las cuatro reglas del *HorarioValidator*, obtiene o crea el *plan_generado* y persiste con *commit*; las excepciones de dominio se propagan sin capturarse en esta capa.

• Contenedor de Inyección de Dependencias. Conforme al argumento de Özkan et al. (2025) sobre la conveniencia de soluciones simples frente a herramientas externas en organizaciones medianas, la clase Container implementa un Service Locator minimalista sin frameworks externos, con dos ámbitos: singleton de sesión para *HorarioService* y *prototype* para validadores sin estado. El adaptador *_HorarioServiceRepoAdapter* permite que los casos de uso operen contra *IHorarioRepository* mientras el repositorio MySQL migra progresivamente hacia dicha interfaz.

• Estado de Sesión. En línea con el principio de inversión de dependencias hacia el dominio de Santiago-Salazar y Rico-Bautista (2024), *DetallePlanState* centraliza el estado mutable de edición, incluido *ids_sesion_por_contexto* (indexado por LIES y semestre), y almacena catálogos en memoria de aulas, docentes y unidades de aprendizaje. Toda validación de reglas de negocio se realiza en el backend, nunca en el estado de sesión.

Implementación de la Capa de Infraestructura. • Modelo de Base de Datos. Retomando las ventajas de MySQL para instituciones con recursos limitados descritas por Wahyudi et al. (2022), la base de datos contiene 15 tablas normalizadas hasta 3FN, en tres grupos: (1) estructura curricular: *nivel_academico*, *plan_estudios*, *semestres*, *detalle_semestre*, *tipo_materia*, *lies_horarios* y la tabla puente *plan_lies*; (2) catálogo de recursos: *materias_tronco*, *optativas*, *asignacion_materia*, *docentes y aulas*; y (3) planificación operativa: *periodo_escolar*, *plan_generado*, *horarios* y *detalle_horario*. Una restricción CHECK en *asignacion_materia* garantiza que cada asignación referencie exactamente una materia de tronco común o una optativa, nunca ambas ni ninguna; los índices de detalle_horario están optimizados para las consultas de traslape sobre *(id_plan_generado, dia, hora_inicio, hora_fin*). Thirtoft (2024) desarrolla python-dotenv, biblioteca mediante la cual las credenciales de MySQL se reciben exclusivamente por variables de entorno.

• Conexión y ORM. SQLAlchemy (2024) fue seleccionado por su madurez como toolkit SQL y Object Relational Mapper(ORM), su soporte del patrón *Repository* mediante sesiones controladas y su compatibilidad con MySQL vía *PyMySQL* sin dependencias de compiladores; Percival y Gregory (2020) sostienen que el patrón *Repository* mantiene el dominio independiente de la tecnología de persistencia; el sistema encapsula todas las consultas dentro de *HorarioRepository*, *PlanesRepository* y *PlanEstudiosRepository*. MySQL se prefirió sobre PostgreSQL y SQLite por su disponibilidad institucional. *DatabaseConnection* implementa Singleton con *pool_pre_ping=True*, y el driver *PyMySQL* simplifica la instalación en Windows sin compiladores.

Implementación de la Capa de Presentación (Flet). La interfaz se construyó con Flet, que expone los widgets de Flutter como clases Python. Su elección se apoya en tres ventajas: un modelo de componentes reactivo que facilita separar presentación de lógica de negocio; ejecución como aplicación nativa de escritorio sin servidor HTTP ni navegador; y tiempos de renderizado inferiores a 16 ms (Flet, 2024).

Siguiendo la separación de capas con dependencias unidireccionales que describen Santiago-Salazar y Rico-Bautista (2024), el sistema es Modelo-Vista-Controlador (MVC); las vistas solo renderizan y capturan eventos; los controladores son responsables de la presentación. El sistema tiene componentes reutilizables como *TablaHorarios, SelectorDocente, BuscadorUnidad, ScrollTimePicker, DialogConflictos, DropdownConNuevo, FilaHorario y ToolbarPlan,* además de tokens de diseño centralizados (Colores, Fuentes).

Implementación de la Generación de PDF. El módulo *PDFGenerator* utiliza la biblioteca de código abierto ReportLab (2024) para generar documentos en formato carta, mediante primitivas gráficas de bajo nivel que permiten construir un horario continuo con celdas de altura proporcional a la duración de la sesión y una paleta de ocho colores pastel. El *GestorMembrete* gestiona la imagen institucional personalizada sin almacenarla en la base de datos y la proporciona como un fondo de página completa. La clase *GeneradorPDFDocente* genera una versión para cada profesor basada en su horario semanal.

Fase 4: Pruebas y Validación. Percival y Gregory (2020) sugieren la pirámide de pruebas que se siguen en el proyecto: pruebas unitarias del dominio en aislamiento con mocks de repositorio, pruebas de integración entre repositorios de SQLAlchemy y base de datos de prueba MySQL (fixtures en *conftest.py*), y pruebas de fuerza bruta que toman todo el proceso desde la creación del plan hasta la exportación del PDF. Conforme al hallazgo de Lukasczyk et al. (2023) sobre la mejora de cobertura al ampliar deliberadamente los escenarios de prueba, un archivo dedicado contiene más de 20 escenarios adicionales para las reglas *cross-LIES* desarrollados con Pytest (Krekel et al., 2024), incluyendo casos de borde: asignación de la misma materia de tronco común en múltiples LIES con docente diferente (debe rechazarse), asignación idéntica en todas las LIES (debe aceptarse) y traslape de optativas en LIES distintas (debe aceptarse).

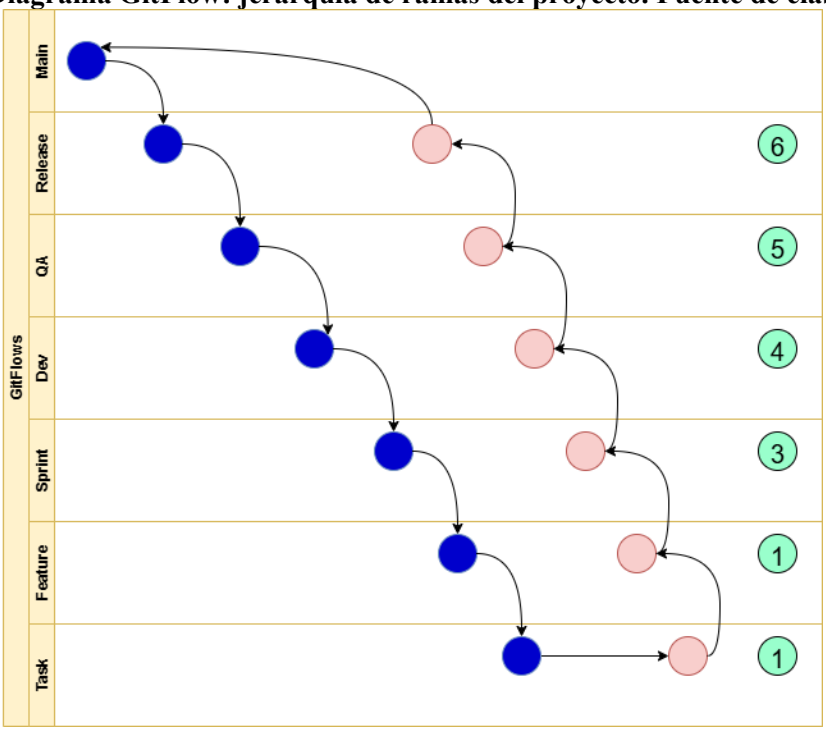
Fase 5: Integración y Entrega – Estrategia de Ramificación GitFlow. Cortés Ríos et al. (2022), en su marco unificador para el análisis sistemático de flujos de trabajo de Git, describen GitFlow como un modelo de ramificación que aísla el trabajo de nuevas funcionalidades del código en producción mediante ramas con propósitos específicos y reglas claras de fusión; el control de versiones del proyecto se gestionó bajo este modelo. La jerarquía de ramas fue: *main* ← *release* ← *qa* ← *dev* ← *sprint* ← *feature* ← *task*, tal y como se enlista en la tabla 5 y se visualiza en la figura 4.

Tabla 5. Jerarquía de ramas de la estrategia GitFlow.

Rama	Propósito	Origen	Destino de fusión
main	Versión estable en producción; solo recibe fusiones desde release	N/A	N/A
release	Preparación de versión; pruebas finales antes de producción	dev	main + dev
qa	Entorno de pruebas de calidad; valida funcionalidades integradas	dev	release
dev	Rama principal de desarrollo; integra las features completadas	main	qa / release
sprint	Agrupa las features desarrolladas durante un sprint de Scrum	dev	dev
feature/*	Desarrollo de una historia de usuario derivada de una épica	sprint	sprint
task/*	Tarea atómica dentro de una feature (ajustes, correcciones)	feature	feature

Fuente de elaboración propia

Figura 4: Diagrama GitFlow: jerarquía de ramas del proyecto. Fuente de elaboración propia



Siguiendo la práctica de gestión segura de credenciales mediante variables de entorno que describe Thirtoft (2024), el repositorio excluye del control de versiones los entornos virtuales, las credenciales del archivo *.env*, la caché de pruebas y los PDF generados en tiempo de ejecución.

RESULTADOS Y DISCUSIÓN

Esta sección presenta los resultados de la implementación y validación del sistema en cuatro apartados: primero, las funcionalidades implementadas y su recorrido por las pantallas principales; luego, se reportan los resultados de las pruebas automatizadas que validan el correcto funcionamiento del motor de reglas y los flujos completos; en tercer lugar, se compara el impacto operativo esperado con el proceso manual anterior; y por último, se presentan las métricas generales del proyecto en términos de tamaño y organización del código. Se comienza con las funcionalidades implementadas, ya que son la manifestación visible de las decisiones arquitectónicas y de dominio descritas en la sección de metodología.

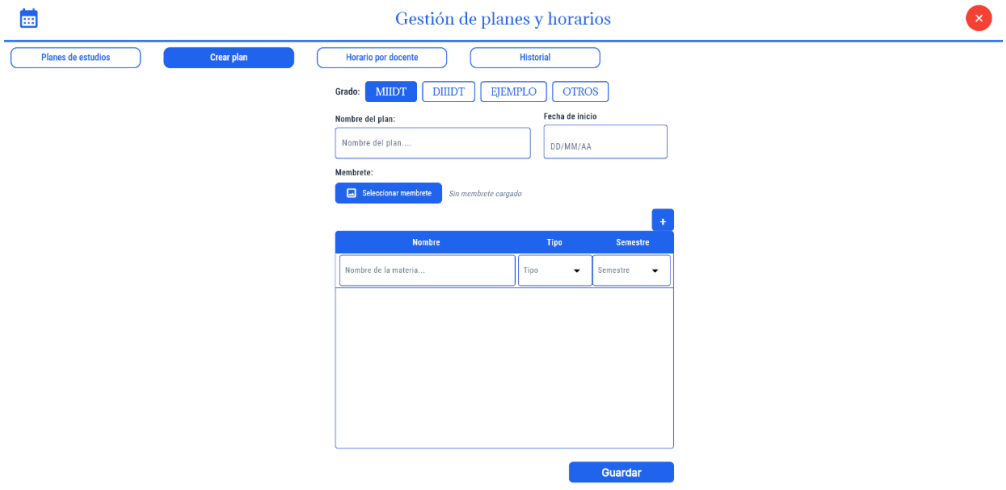
Funcionalidades Implementadas. El sistema implementado cubre el ciclo de gestión académica del posgrado a través de cuatro pestañas principales accesibles desde la vista de Planes.

La pestaña Planes de estudios (figura 5) presenta el catálogo de planes activos e inactivos, con selector de nivel académico y selector de plan. La animación de puerta señaliza la navegación hacia la vista de detalle del plan seleccionado.

Figura 5: Pantalla principal, punto de acceso a los módulos. Fuente de elaboración propia

La figura 6 presenta la pestaña Crear plan, donde el usuario registra un nuevo plan de estudios indicando el grado, el nombre del plan, la fecha de inicio, el membrete institucional y las materias, especificando para cada una su tipo y semestre correspondiente.

Figura 6: Crear plan: registro de nuevo plan de estudios. Fuente de elaboración propia



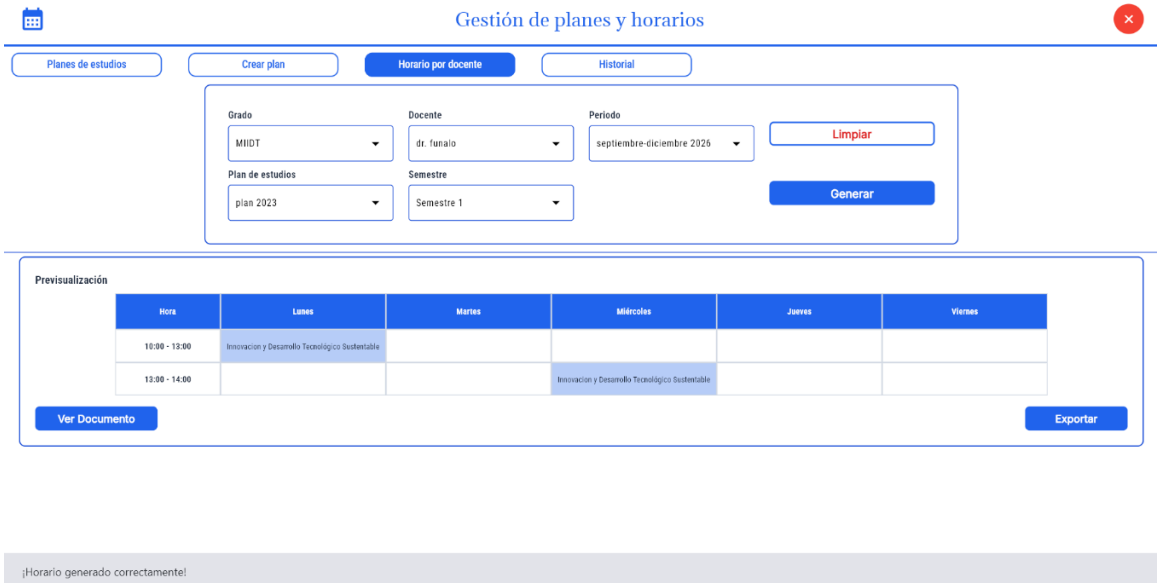
La vista Detalle de plan que se muestra en la figura 7 es la más compleja del sistema. Presenta una tabla editable de horarios para la LIES y el semestre activo, con selectores de docente, aula, día, hora de inicio y hora de fin (mediante el componente *ScrollTimePicker*). Al confirmar una fila, el controlador invoca

el caso de uso correspondiente (crear, editar o eliminar) y, si se detecta un conflicto, se activa el *DialogConflictos* el cual despliega la descripción completa del tipo de restricción violada.

Figura 7: Generador de horarios: permite crear un horario correspondiente. Fuente de elaboración propia

La pestaña Horario por docente (figura 8) muestra la carga horaria semanal de cualquier docente filtrada según grado, período, semestre y plan, y permite exportar el comprobante individual en PDF.

Figura 8: Horario por docente: generación automática del horario de un docente. Fuente de elaboración propia.



La pestaña Historial expuesta en la figura 9, presenta los planes generados por período con cascada de filtros *Grado → Período → Plan de estudios → Semestre*, y permite acceder al horario de cualquier período anterior para su eliminación o reexportación.

Figura 9: Historial: consulta de horarios ya generados. Fuente de elaboración propia.

Antes de exportar, en la figura 10 el sistema muestra una vista previa del documento final con el membrete institucional, la distribución semanal del horario y el listado de asignaturas.

Figura 10: Vista previa antes de exportar el documento. Fuente de elaboración propia.

Resultados de las Pruebas Automatizadas. Según la pirámide de pruebas descrita por Percival y Gregory (2020) y aplicada en la fase de Pruebas y Validación, la Tabla 6 muestra los resultados de menor a mayor nivel de detalle: primero pruebas unitarias solo en el dominio, luego pruebas de integración entre los repositorios y la base de datos, seguidas de pruebas de extremo a extremo en todo

el flujo del sistema y finalmente escenarios de validación cruzada LIES que utilizan específicamente la Regla 1 del *HoarioValidator*. Para cada nivel, se informa el número de casos implementados, las proporciones de casos exitosos y el alcance funcional cubierto para que se pueda verificar que la cobertura de pruebas escala con la complejidad de la lógica de negocio validada.

Tabla 6. Resumen de pruebas automatizadas con Pytest.

Nivel de prueba	Casos impl.	Exitosos	Ámbito cubierto
Unitarias (Dominio)	47	47 (100 %)	Entidades, *Value Objects, Validator* (4 reglas), Caso de Uso Crear
Integración (Repositorios)	18	18 (100 %)	*HorarioRepository, PlanesRepository* con BD MySQL de prueba
Extremo a extremo	12	12 (100 %)	Flujo completo: *crear plan → asignar horario → detectar conflicto → exportar PDF*
Validación cross-LIES (tronco)	> 20	100 %	Casos de borde Regla 1: coincidencia exacta, docente diferente, aula diferente
TOTAL	≥ 97	100 %	Cobertura de todos los escenarios críticos de negocio

Fuente de elaboración propia.

Impacto Operativo (esperado). El sistema aún no se ha puesto en operación durante un ciclo escolar completo. Los valores presentados en la Tabla 7 corresponden a la mejora esperada, calculada a partir de la comparación entre el proceso manual actual y el tiempo observado del sistema en pruebas controladas; la validación en un ciclo real será parte del trabajo futuro.

Tabla 7. Comparación entre el proceso manual actual y el desempeño esperado del sistema.

Indicador	Proceso manual (antes)	Sistema automatizado (esperado)
Tiempo elaboración horario completo	3–5 días hábiles	2–4 horas (reducción esperada ~90 %)
Detección de traslapes	Manual, revisión por 2 personas (1 día)	Automática, tiempo real (< 500 ms)
Generación de PDF de horario	Manual, diseño en procesador de textos	< 3 segundos, automático
Consistencia plan–horario	Sin garantía (proceso independiente)	Garantizada por la arquitectura
Historial de períodos anteriores	Archivos dispersos, sin estructura	Centralizado, consultable por cascada de filtros

Fuente de elaboración propia.

El proyecto consta de 113 archivos Python propios del sistema (excluyendo el entorno virtual), distribuidos en 4 capas y 22 módulos (figura 11). Los módulos de mayor complejidad son *horario_service.py*, *detalle_plan_view.py* y *horario_repository.py*. El repositorio Git registra 20 ramas de tarea con nombres *Task-DD-MM-AAAA*, integrando ramas Feature y QA en paralelo, y mantiene cero warnings de importación cruzada entre capas a lo largo de los seis sprint.

Figura 11: Estructura de directorios del proyecto, organizada por capas (Presentación, Aplicación, Dominio, Infraestructura). Fuente de elaboración propia.

Comparación con Sistemas Similares. Frente al proceso manual previo, el sistema está diseñado para eliminar los errores de traslape mediante validación automática en cuatro capas de restricción, con una reducción esperada del tiempo administrativo cercana al 90%. A diferencia de soluciones genéricas como FET (Lalescu, 2024), el sistema fue diseñado específicamente para satisfacer las especificidades del programa de posgrado MIIDT (gestión de LIES, distinción entre tronco común/optativa, lógica cruzada de LIES (*cross-LIES*) y membrete institucional) sin configuración compleja.

En contraste con los sistemas reportados por García Yáñez et al. (2024) y Romaguera et al. (2024), el sistema se basa en una lógica única y universal. En el sistema la implementación de DDD (especificaciones combinables, reglas de entidad documentadas, excepciones de dominio con semántica precisa) es más clara y robusta para reglas de negocio complicadas. A diferencia de los sistemas web, la solución es una aplicación de escritorio, por lo que el acceso está restringido al control remoto sin necesidad de utilizar ninguna red o entorno adicional.

Limitaciones Actuales. El sistema tiene cuatro limitaciones según se evaluó esto, a saber: (1) la base de datos se ejecuta en localhost y se necesita usar un servidor MySQL compartido para acceder a ella desde diferentes dispositivos al mismo tiempo, (2) no implementa autenticación basada en roles donde

cualquier usuario tiene acceso a la base de datos y puede cambiar los horarios, (3) el motor de validación solo funciona con las reglas en el código y no se agregan más restricciones desde la interfaz, y (4) el sistema permite la generación manual de los horarios validados pero no genera automáticamente el horario basado en metaheurísticas.

Trabajo Futuro. Las mejoras incluyen la migración del sistema a una arquitectura web basada en FastAPI y React o Next.js utilizando la interfaz *IHorarioRepository* ya definida; la adición de un módulo de autenticación con JWT para coordinadores, personal administrativo y docentes; la integración de algoritmos genéticos y búsqueda tabú para la generación automática de horarios, como lo propuesto por Bashab et al. (2023), con el SIAE de UAGro a través de APIs REST; y el desarrollo de un módulo de informes estadísticos para monitorear la carga docente y el uso de aulas utilizando la estructura de las quince tablas normalizadas. Dado que esta integración expondría datos académicos sensibles a través de una API REST, dicho desarrollo debe tener en cuenta los riesgos de seguridad y la herramienta de monitoreo basada en el enfoque orientado a datos propuesto por Valencia (2025) para la misma coordinación. Específicamente, detecta el riesgo API4:2023 - Consumo de Recursos Sin Restricciones del top 10 de OWASP y genera alertas en caso de fugas de datos.

CONCLUSIONES

El sistema de gestión de planes académicos y horarios para la Coordinación de Posgrado en Ingeniería de la MIIDT cumple con los objetivos. La aplicación construida en Python y Flet muestra cómo se puede construir software académico con rigor arquitectónico (cuatro capas, Diseño Orientado al Dominio, inyección de dependencias, patrón de Repositorio) en código abierto y sin costos de licencia.

La separación entre el núcleo del dominio y la infraestructura se verifica en cada revisión de sprint y las cuatro reglas de validación de horarios pueden cambiarse o ampliarse sin afectar las capas de presentación e infraestructura, lo cual es importante en un contexto donde las reglas de negocio cambian con cada actualización de plan de estudios o la estructura del LIES.

Se estima una reducción de casi el 90% en el tiempo de preparación de horarios, una latencia de validación de menos de 500 ms y una lista de 77 pruebas automatizadas exitosas. El proyecto está disponible bajo la licencia MIT (Open Source Initiative, s.f.), una de las licencias permisivas más populares en el software académico y de código abierto, debido a sus requisitos mínimos de

cumplimiento (mantener el aviso de derechos de autor y permiso), el uso, copia, modificación y redistribución del código sin restricciones adicionales; esto facilita que otras coordinaciones de posgrado en universidades públicas adopten, adapten y reutilicen el sistema como un estudio de caso sin incurrir en costos de licencias.

REFERENCIAS BIBLIOGRÁFICAS

Akour, M., & Alenezi, M. (2022). Higher education future in the era of digital transformation. Education Sciences, 12(11), 784. https://doi.org/10.3390/educsci12110784

Bashab, A., Ibrahim, A. O., Hashem, I. A. T., Aggarwal, K., Mukhlif, F., Ghaleb, F. A., & Abdelmaboud, A. (2023). Optimization techniques in university timetabling problem: Constraints, methodologies, benchmarks, and open issues. Computers, Materials & Continua, 74(3), 6461–6484. https://doi.org/10.32604/cmc.2023.034051

Ceschia, S., Di Gaspero, L., & Schaerf, A. (2023). Educational timetabling: Problems, benchmarks, and state-of-the-art results. European Journal of Operational Research, 308(1), 1–18. https://doi.org/10.1016/j.ejor.2022.07.011

Cortés Ríos, J. C., Embury, S. M., & Eraslan, S. (2022). A unifying framework for the systematic analysis of Git workflows. Information and Software Technology, 145, 106811. https://doi.org/10.1016/j.infsof.2021.106811

Flet. (2024). Flet: Build multi-platform apps in Python powered by Flutter (Version 0.28) [Software]. https://flet.dev

García Yáñez, J. A., Ramírez Sáenz, F. D. J., Ríos Saldaña, E. L., Rubio Aguilar, M. A., Barrios Flores, I. E., García Flores, C. L., Hernández Cerda, R. E., & Zacarías Ortiz, J. J. (2024). Development of a web-based timetabling software for a Mexican university. Working Paper WP-2024-006, Center for the Study of Western Hemispheric Trade, Texas A&M International University. https://www.tamiu.edu/cswht/documents/wp-2024-006-garcia-yanez.pdf

Krekel, H., Oliveira, B., Pfannschmidt, R., Bruynooghe, F., Laugher, B., & Bruhin, F. (2024). Pytest: Helps you write better programs (Version 8.x) [Software]. https://docs.pytest.org

Lalescu, L. (2024). FET: Free Timetabling Software [Software]. https://lalescu.ro/liviu/fet/

Lima, C. R. C., Carr, C. N., Margarido, J. J. P., & Silva, R. D. (2023). O modelo incremental no desenvolvimento de software: Uma maneira estruturada e interativa de entregar produtos de qualidade. Research, Society and Development, 12(4), e7512440934. https://doi.org/10.33448/rsd-v12i4.40934

Liu, M., Huang, X., He, W., Xie, Y., Zhang, J. M., Jing, X., Chen, Z., & Ma, Y. (2024). Research artifacts in software engineering publications: Status and trends. Journal of Systems and Software, 213, 112032. https://doi.org/10.1016/j.jss.2024.112032

Lukasczyk, S., Kroiß, F., & Fraser, G. (2023). An empirical study of automated unit test generation for Python. Empirical Software Engineering, 28(2), 36. https://doi.org/10.1007/s10664-022-10248-w

Open Source Initiative. (s.f.). The MIT License. https://opensource.org/license/mit

Özkan, O., Babur, Ö., & van den Brand, M. (2025). Domain-Driven Design in software development: A systematic literature review on implementation, challenges, and effectiveness. Journal of Systems and Software, 230, 112537. https://doi.org/10.1016/j.jss.2025.112537

Percival, H., & Gregory, B. (2020). Architecture patterns with Python: Enabling test-driven development, domain-driven design, and event-driven microservices. O'Reilly Media.

Pressman, R. S., & Maxim, B. R. (2021). Ingeniería del software: Un enfoque práctico (9.ª ed.). McGraw-Hill Education.

Python Software Foundation. (2024). What's new in Python 3.13. https://docs.python.org/3/whatsnew/3.13.html

ReportLab. (2024). ReportLab open-source PDF library (Version 4.x) [Software]. https://www.reportlab.com/opensource/

Romaguera, D., Plender-Nabas, J., Matias, J., & Austero, L. (2024). Development of a web-based course timetabling system based on an enhanced genetic algorithm. Procedia Computer Science, 234, 1714–1721. https://doi.org/10.1016/j.procs.2024.03.177

Russell, S. J., & Norvig, P. (2020). Artificial Intelligence: A Modern Approach (4.ª ed.). Pearson.

Sangabriel-Alarcón, J., Ocharán-Hernández, J. O., Cortés-Verdín, K., & Limón, X. (2024). Domain-Driven Design in microservices-based systems development: A systematic literature review and

thematic analysis. Programming and Computer Software, 50(8), 742–770. https://doi.org/10.1134/S0361768824700749

Santiago-Salazar, J. A., & Rico-Bautista, D. (2024). Clean Architecture: Impact on Performance and Maintainability of Native Android Projects. In Advances in Computing (CCC 2023), Communications in Computer and Information Science, vol. 1924. Springer, Cham. https://doi.org/10.1007/978-3-031-47372-2_8

Schwaber, K., & Sutherland, J. (2020). The Scrum Guide: The definitive guide to Scrum: The rules of the game. Scrum.org. https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf

SQLAlchemy. (2024). SQLAlchemy 2.0: The Python SQL toolkit and Object Relational Mapper [Software]. https://docs.sqlalchemy.org/en/20/

Thirtoft, J. (2024). python-dotenv: Read key-value pairs from a .env file (Version 1.0) [Software]. https://github.com/theskumar/python-dotenv

Useche, A. C., Galvis, Á. H., Díaz-Barriga Arceo, F., Patiño Rivera, A. E., & Muñoz-Reyes, C. (2022). Reflexive pedagogy at the heart of educational digital transformation in Latin American higher education institutions. International Journal of Educational Technology in Higher Education, 19, 62. https://doi.org/10.1186/s41239-022-00365-3

Valencia, R. E. C., Villegas, A. C., Morales, A. F., & Carmona, C. Á. (2025). Monitoring Software Tool to Prevent Data Leaks in a RESTful API. In R. Valencia-García, T. Borodulina, J. Del Cioppo-Morstadt, C. E. Moran-Castro, & N. Vera-Lucio (Eds.), Technologies and Innovation. CITI 2024, Communications in Computer and Information Science, vol. 2276. Springer, Cham. https://doi.org/10.1007/978-3-031-75702-0_14

Wahyudi, J., Asbari, M., Sasono, I., Pramono, T., & Novitasari, D. (2022). Database Management Education in MYSQL. Edumaspul: Jurnal Pendidikan, 6(2), 2413–2417. https://doi.org/10.33487/edumaspul.v6i2.4570

Zhong, C., Li, S., Huang, H., Liu, X., Chen, Z., Zhang, Y., & Zhang, H. (2024). Domain-Driven Design for microservices: An evidence-based investigation. IEEE Transactions on Software Engineering, 50(6), 1425–1449. https://doi.org/10.1109/TSE.2024.3385835