Universidad Continental

# FINPROC: SIMULADOR DE PROCESOS DE ATENCIÓN FINANCIERA BANCARIA

**Informe Técnico – Entregable 1: Planificación y Diseño**

*Curso: Estructura de Datos*
*Docente: Yesenia Concha Ramos*
*Integrantes:*


- *Jhon gustavo Ccarita Velasquez (100%)*
- *Rodrigo Sevillanos Tinco(100%)*
- *Andre Sebastian  Espinosa Zea(100%)*


*Tema: Entregable 1*
*NRC: 59098*

2025-02
Cusco-Perú

# Índice General

## Formación de equipos y distribución de roles

El equipo de desarrollo del proyecto FINPROC: Simulador de Procesos de Atención Financiera Bancaria está conformado por tres integrantes, quienes asumen funciones específicas de acuerdo con sus habilidades y responsabilidades dentro del proceso de desarrollo.

**Integrantes y roles asignados:**

1. **Gustavo Jhon Ccarita Velásquez – Scrum Master y desarrollador principal**

   Responsable de dirigir la planificación y asignación de tareas mediante la metodología ágil **Scrum**, utilizando **Obsidian** con los complementos *Kanban* y *Tasks* para la gestión visual del flujo de trabajo. Supervisa el cumplimiento de los objetivos semanales, fomenta la colaboración y participa activamente en el desarrollo del código y la integración de los módulos del sistema.

2. **Rodrigo Sevillanos Tinco – Analista y diseñador de algoritmos**

   Encargado de realizar el análisis funcional del sistema, diseñar los diagramas de flujo y elaborar los algoritmos principales asociados a las estructuras de datos (lista enlazada, cola y pila). Participa en la documentación técnica y asegura la coherencia entre el diseño lógico y la implementación del sistema.

3. **André Sebastián Espinoza Zea – Tester y responsable de documentación**

   Responsable de la validación funcional del sistema, realizando pruebas exhaustivas de cada módulo. Además, gestiona la documentación técnica, recopila las evidencias del trabajo colaborativo y mantiene actualizado el informe del proyecto, incluyendo avances, actas de reunión y resultados de validación.

**Creación del repositorio para el desarrollo del trabajo:**

https://github.com/HustavoJhon/FinProc

## Planificación de tareas y responsabilidades por cada integrantes

### Metodología de Trabajo: Scrum Adaptado

El equipo adoptará una **metodología ágil basada en Scrum**, ajustada a la escala académica del proyecto.

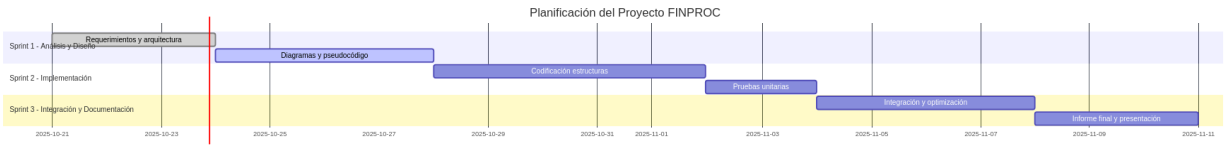
El desarrollo se organizará en **tres sprints semanales**, cada uno con un conjunto definido de objetivos y tareas.

- **Sprint Planning (día 1):** reunión inicial de cada semana para asignar tareas y estimar tiempos.
- **Daily Scrum:** reuniones breves 2 veces por semana (lunes y jueves) para revisar avances y obstáculos.
- **Sprint Review (día 7):** revisión del trabajo finalizado y documentación de resultados.
- **Sprint Retrospective:** ajustes de metodología y redistribución de tareas según desempeño.

**Adaptado Planificación de Sprints (Gantt)**

| semana | sprints | Objetivos Principales | Tareas Clave | Reuniones |
|---|---|---|---|---|
| Semana 1 (Del 21 al 27 oct) | Sprint 1: Análisis y diseño | Definir requerimientos y arquitectura del sistema | Definir estructura, elaborará diagrama y pseudocódigo | Lunes(Planificación), Jueves (daily |
| Semana 2 (Del 28 oct al 3 nov) | Sprint 2: Implementación | Codificación de estructura (lista, pila, cola) y pruebas unitarias | Programacion en C++, validación de inserciones, eliminación y búsqueda búsquedas | Lunes (daily), Viernes (review) |
| Semana 3 (Del 4 al 10 nov) | Spirit 3: Integración y Documentación | Integrar módulos, optimización, documentar y preparar presentación | Ensayo de sustentación, commits finales y revisión del informe | Martes (daily) , Domingo (retrospectiva) |

*(Figura: Diagrama Gantt de los sprints)*




# CAPÍTULO 1

## 1.-Análisis del Problema

### 1.1. Descripción del problema

En las entidades financieras, la atención de clientes suele presentar problemas de demora y desorganización, especialmente cuando se manejan múltiples tipos de operaciones como depósitos, retiros o solicitudes de préstamos. El objetivo de este proyecto es simular el funcionamiento de un sistema de atención bancaria que gestione de forma ordenada los procesos de atención y registro de transacciones, aplicando estructuras de datos dinámicas lineales (listas enlazadas, colas y pilas) para mejorar la eficiencia y organización.

### 1.2. Requerimientos del sistema

**Requerimientos funcionales**

- Registrar nuevos clientes y almacenarlos en una lista enlazada.
- Gestionar una cola de atención priorizada según el tipo de cliente o servicio.
- Registrar las transacciones realizadas (retiros, depósitos, préstamos).

**Diagramas de las estructuras de datos a implementar**

- Guardar y cargar los datos desde archivos (persistencia de información).
- Permitir visualizar el estado actual de los procesos y la memoria.

**Requerimientos no funcionales**

- El sistema debe ser desarrollado en Dev-C++ y ejecutarse por consola.
- El código debe estar modularizado, comentado y documentado.

- Se debe garantizar una ejecución eficiente en tiempo y memoria.
- Interfaz sencilla, amigable y clara para el usuario.

### 1.3. Estructuras de datos propuestas

- **Lista enlazada:** para el registro general de clientes.
- **Cola:** para la atención de clientes según prioridad.
- **Pila:** para el historial de transacciones y operaciones realizadas.

### 1.4. Justificación de la elección

El uso de estas estructuras permite modelar de manera eficiente el comportamiento real de un sistema de atención financiera. Las listas enlazadas ofrecen flexibilidad en el manejo dinámico de clientes, las colas permiten la organización de la atención en orden de llegada o prioridad, y las pilas facilitan la gestión del historial y la posibilidad de deshacer operaciones

# CAPÍTULO 2

---

## Diseño de la Solución

---

### 2.1. Descripción general del diseño

El sistema se dividirá en tres módulos principales:

1. **Gestor de Clientes (Lista Enlazada):**

   Permite agregar, eliminar y buscar clientes.

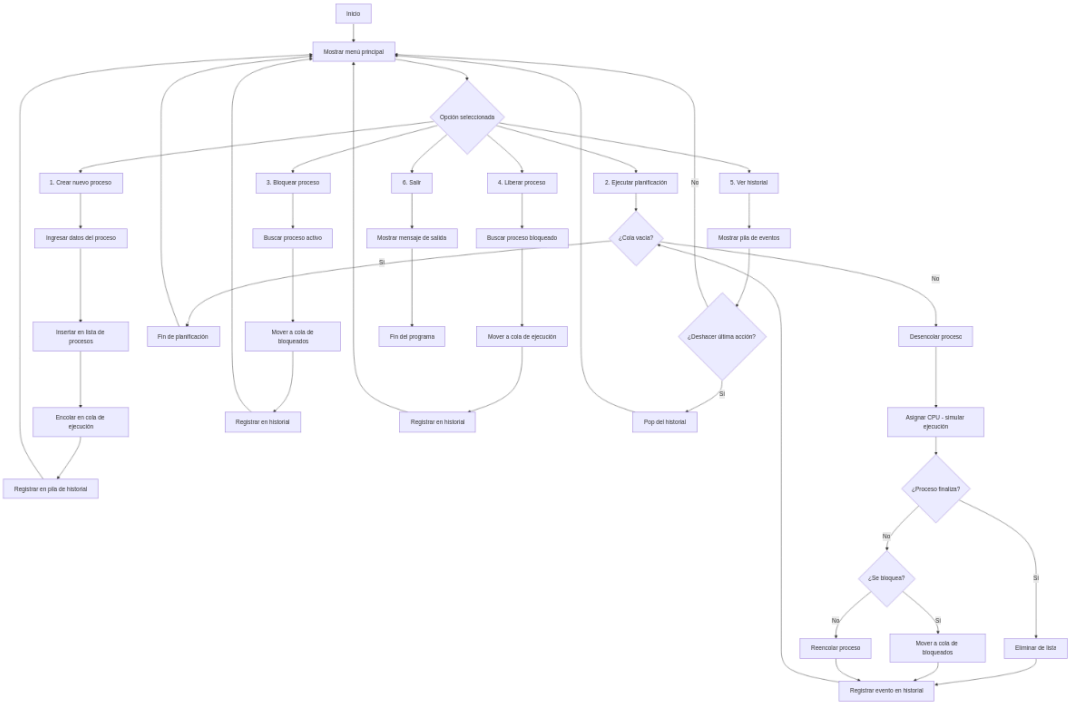
2. **Gestor de Atención (Cola de Prioridad):**

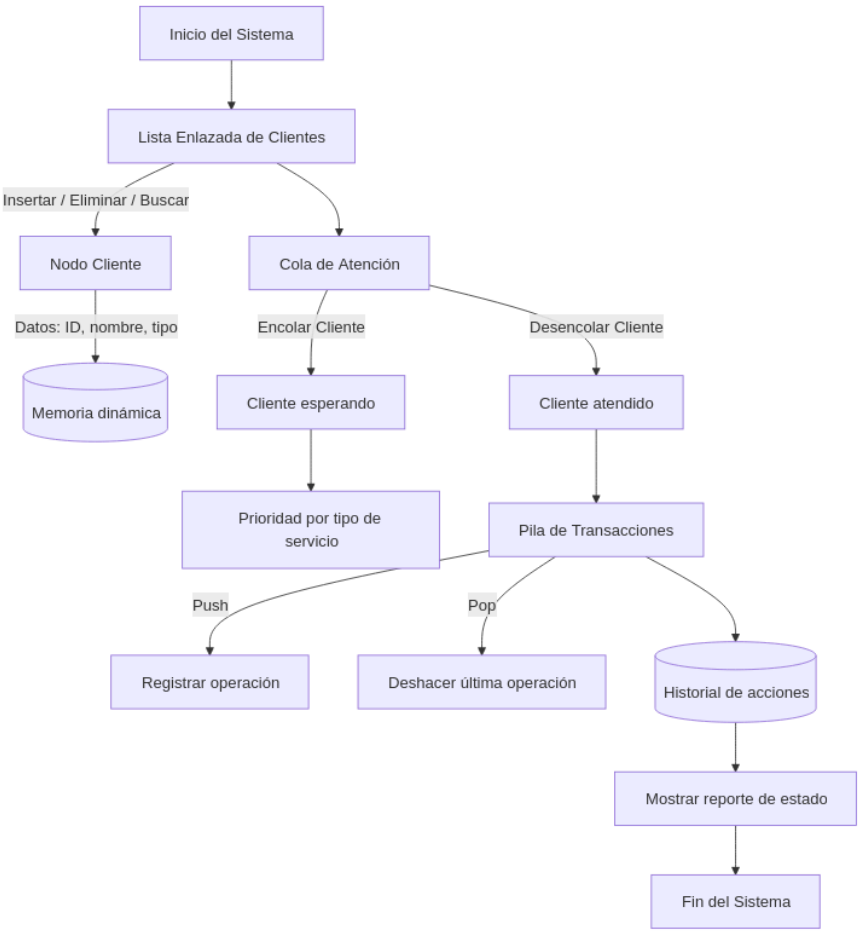
   Controla el orden de atención de los clientes en ventanilla.

3. **Gestor de Transacciones (Pila):**

   Almacena las operaciones realizadas por cada cliente.

**Diagramas de las estructuras de datos a implementar**


Inicio
Mostrar menú principal
Opción seleccionada
1. Crear nuevo proceso
Ingresar datos del proceso
Insertar en lista de procesos
Encolar en cola de ejecución
Registrar en pila de historial
3. Bloquear proceso
Buscar proceso activo
Mover a cola de bloqueados
Registrar en historial
6. Salir
Mostrar mensaje de salida
Fin del programa
Registrar en historial
4. Liberar proceso
Buscar proceso bloqueado
Mover a cola de ejecución
2. Ejecutar planificación
¿Cola vacía?
Fin de planificación
¿Deshacer última acción?
Pop del historial
5. Ver historial
Mostrar pila de eventos
Desencolar proceso
Asignar CPU - simular ejecución
¿Proceso finaliza?
¿Se bloquea?
Reencolar proceso
Mover a cola de bloqueados
Eliminar de lista
Registrar evento en historial
Sí
No


Inicio del Sistema
Lista Enlazada de Clientes
Insertar / Eliminar / Buscar
Nodo Cliente
Cola de Atención
Datos: ID, nombre, tipo
Encolar Cliente
Desencolar Cliente
Memoria dinámica
Cliente esperando
Cliente atendido
Prioridad por tipo de servicio
Pila de Transacciones
Push
Pop
Registrar operación
Deshacer última operación
Historial de acciones
Mostrar reporte de estado
Fin del Sistema

## 2.2. Algoritmos principales

**Pseudocódigo:**

**Algoritmo 1: Agregar proceso (cliente nuevo)**

```
ALGORITMO AgregarCliente
// Inserta un nuevo cliente en la lista enlazada de clientes

ENTRADAS:
    datosCliente ← información del nuevo cliente

VARIABLES:
    nuevoNodo, actual

INICIO
    nuevoNodo ← CrearNodo(datosCliente)

    SI (listaClientes está vacía) ENTONCES
        listaClientes.inicio ← nuevoNodo
    SINO
        actual ← listaClientes.inicio
        MIENTRAS (actual.siguiente ≠ NULO) HACER
            actual ← actual.siguiente
        FIN MIENTRAS
        actual.siguiente ← nuevoNodo
    FIN SI

    MOSTRAR("Cliente agregado correctamente")
FIN
```

**Algoritmo 2: Cambiar estado de atención**

```
ALGORITMO AtenderCliente
// Desencola un cliente y registra su atención

VARIABLES:
    clienteActual

INICIO
    SI (colaAtencion está vacía) ENTONCES
        MOSTRAR("No hay clientes en espera")
    SINO
        clienteActual ← Desencolar(colaAtencion)
        MOSTRAR("Atendiendo a: ", clienteActual.nombre)

        // Registrar transacción en la pila de historial
        Push(pilaHistorial, clienteActual.operacion)

        MOSTRAR("Operación registrada con éxito")
    FIN SI
FIN
```

**Algoritmo 3: Registrar transacción**

```
ALGORITMO RegistrarTransaccion
// Guarda la operación realizada por un cliente en la pila

ENTRADAS:
    idCliente, tipoOperacion, monto

VARIABLES:
    nuevaTransaccion

INICIO
    nuevaTransaccion ← CrearTransaccion(idCliente, tipoOperacion, monto)
    Push(pilaTransacciones, nuevaTransaccion)
    MOSTRAR("Transacción registrada para el cliente ", idCliente)
FIN
```

## 2.3. Justificación del diseño

El diseño modular facilita la comprensión y mantenimiento del sistema. El uso de estructuras dinámicas mejora la eficiencia, reduciendo la complejidad de inserción y eliminación de datos. Además, el enfoque simula con fidelidad los procesos de atención reales en una entidad bancaria.