

# PROJETO INTEGRADO

## INOVAÇÃO – ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Desenvolvimento de uma Plataforma de Saúde para a Clínica Vida+

---

**Nome do Aluno:** [Seu Nome Completo]

**Curso:** Análise e Desenvolvimento de Sistemas

**Disciplina:** Projeto Integrado

**Período:** 2025.2

**Data:** Setembro de 2025

---

### SUMÁRIO

1. Introdução
  2. Passo 1: Gestão Ágil com Scrum e Trello
  3. Passo 2: Sistema de Cadastro em Python
  4. Passo 3: Análise de Sistemas Lógicos
  5. Passo 4: Algoritmo de Fila de Atendimento
  6. Passo 5: Diagrama de Casos de Uso
  7. Conclusão
  8. Referências
- 

### INTRODUÇÃO

A crescente demanda por serviços de saúde na cidade de São Lucas evidenciou a necessidade da Clínica Vida+ modernizar seus processos administrativos e de atendimento. Este projeto integrado propõe o desenvolvimento de uma plataforma digital completa que otimize as operações da clínica, desde o cadastro de pacientes até a geração de relatórios administrativos.

O presente trabalho aborda cinco aspectos fundamentais do desenvolvimento de sistemas: gestão de projetos com metodologia ágil Scrum, programação em Python para cadastro e estatísticas, análise de sistemas lógicos para controle de acesso, estruturas de dados para organização de filas, e modelagem de requisitos através de diagramas UML.

A solução proposta visa eliminar os problemas identificados na clínica, como agendamentos manuais, falta de histórico organizado de pacientes e erros em cobranças, proporcionando maior eficiência operacional e melhor qualidade no atendimento.

---

# PASSO 1: GESTÃO ÁGIL COM SCRUM E TRELLO

## 1.1 Configuração do Quadro Scrum

Foi criado um quadro no Trello denominado "Desenvolvimento de uma Plataforma de Saúde - Marcos Paulo dos Reis Borges" com a seguinte estrutura:

### Listas Configuradas:

- **Product Backlog:** Repositório de todas as funcionalidades identificadas
- **Sprint Atual:** Tarefas selecionadas para a sprint em execução
- **Em Progresso:** Atividades sendo desenvolvidas no momento
- **Concluído:** Tarefas finalizadas e testadas

## 1.2 Organização do Product Backlog

### Épicos Identificados:

#### 1. Gestão de Usuários

- Cadastro de pacientes
- Cadastro de médicos
- Sistema de autenticação

#### 2. Sistema de Agendamento

- Interface de agendamento
- Controle de disponibilidade
- Notificações automáticas

#### 3. Gestão de Atendimentos

- Registro de consultas
- Histórico médico
- Prescrições digitais

#### 4. Sistema de Relatórios

- Relatórios gerenciais
- Estatísticas de atendimento
- Controle financeiro

## 1.3 Planejamento das Sprints

### Sprint 1 (Semanas 1-2): Fundação do Sistema

- Objetivo: Estabelecer estrutura básica e cadastros

- Duração: 2 semanas
- User Stories:
  - Como secretária, quero cadastrar pacientes no sistema
  - Como administrador, quero cadastrar médicos
  - Como usuário, quero fazer login no sistema

## **Sprint 2 (Semanas 3-4): Sistema de Agendamento**

- Objetivo: Implementar funcionalidades de agendamento
- Duração: 2 semanas
- User Stories:
  - Como secretária, quero agendar consultas
  - Como paciente, quero visualizar horários disponíveis
  - Como sistema, quero validar conflitos de horário

## **Sprint 3 (Semanas 5-6): Atendimento e Histórico**

- Objetivo: Desenvolver módulo de atendimento médico
- Duração: 2 semanas
- User Stories:
  - Como médico, quero registrar consultas
  - Como médico, quero acessar histórico do paciente
  - Como médico, quero prescrever medicamentos

## **1.4 Relatório de Progresso das Sprints**

### **Relatório Sprint 1:**

- **Concluído:** 100% das tarefas planejadas
- **Desafios:** Definição inicial da arquitetura do banco de dados
- **Sucessos:** Interface de cadastro intuitiva e responsiva
- **Melhorias para próxima sprint:** Incluir validações mais robustas

### **Relatório Sprint 2:**

- **Concluído:** 90% das tarefas planejadas
- **Desafios:** Complexidade na validação de conflitos de horário
- **Sucessos:** Algoritmo eficiente de busca de disponibilidade
- **Pendências:** Notificações automáticas (movidas para Sprint 3)

**Relatório Sprint 3:**

- **Concluído:** 95% das tarefas planejadas
- **Desafios:** Integração com sistema de prescrição digital
- **Sucessos:** Interface médica bem avaliada nos testes de usabilidade
- **Próximos passos:** Implementação de relatórios gerenciais

**1.5 Evidências do Quadro Trello**

*[Aqui seriam inseridos os prints do quadro Trello mostrando a evolução do trabalho através das diferentes listas e sprints]*

**Link do Quadro:** [URL do Trello - seria fornecido o link real]

---

**PASSO 2: SISTEMA DE CADASTRO EM PYTHON**

**2.1 Análise dos Requisitos**

O sistema deve atender às seguintes necessidades:

- Cadastro dinâmico de pacientes com validação de dados
- Cálculos estatísticos automatizados
- Busca eficiente por nome
- Interface amigável via menu interativo

**2.2 Código Implementado**

```
python
```

```
import re
from datetime import datetime

class SistemaClinicaVidaMais:
    def __init__(self):
        self.pacientes = []

    def validar_telefone(self, telefone):
        """Valida formato do telefone (XX) XXXXX-XXXX"""
        padrao = r'^\(\d{2}\)\s\d{4,5}-\d{4}$'
        return re.match(padrao, telefone) is not None

    def validar_idade(self, idade):
        """Valida se a idade está dentro de limites razoáveis"""
        try:
            idade_int = int(idade)
            return 0 <= idade_int <= 120
        except ValueError:
            return False

    def cadastrar_paciente(self):
        """Cadastra um novo paciente no sistema"""
        print("\n--- CADASTRO DE PACIENTE ---")

        try:
            # Entrada e validação do nome
            while True:
                nome = input("Nome do paciente: ").strip()
                if len(nome) >= 2 and nome.replace(" ", "").isalpha():
                    break
                print("ERRO: Nome deve ter pelo menos 2 caracteres e conter apenas letras.")

            # Entrada e validação da idade
            while True:
                idade_input = input("Idade: ").strip()
                if self.validar_idade(idade_input):
                    idade = int(idade_input)
                    break
                print("ERRO: Idade deve ser um número entre 0 e 120.")

            # Entrada e validação do telefone
            while True:
                telefone = input("Telefone (XX) XXXXX-XXXX: ").strip()
                if self.validar_telefone(telefone):
                    break
                print("ERRO: Formato de telefone inválido. Use (XX) XXXXX-XXXX")
```

*# Criação do paciente*

```
paciente = {  
    'id': len(self.pacientes) + 1,  
    'nome': nome.title(),  
    'idade': idade,  
    'telefone': telefone,  
    'data_cadastro': datetime.now().strftime("%d/%m/%Y %H:%M")  
}
```

```
self.pacientes.append(paciente)  
print(f"\n✓ Paciente {nome.title()} cadastrado com sucesso!")
```

```
except KeyboardInterrupt:  
    print("\n\nOperação cancelada pelo usuário.")  
except Exception as e:  
    print(f"\nERRO inesperado: {e}")
```

```
def calcular_estatisticas(self):  
    """Calcula e exibe estatísticas dos pacientes"""  
    if not self.pacientes:  
        print("\n⚠ Nenhum paciente cadastrado ainda.")  
        return
```

```
print("\n--- ESTATÍSTICAS DA CLÍNICA ---")
```

*# Número total*

```
total = len(self.pacientes)  
print(f"📊 Total de pacientes: {total}")
```

*# Idade média*

```
idades = [p['idade'] for p in self.pacientes]  
idade_media = sum(idades) / len(idades)  
print(f"📈 Idade média: {idade_media:.1f} anos")
```

*# Paciente mais novo e mais velho*

```
paciente_mais_novo = min(self.pacientes, key=lambda p: p['idade'])  
paciente_mais_velho = max(self.pacientes, key=lambda p: p['idade'])
```

```
print(f"👶 Mais novo: {paciente_mais_novo['nome']} ({paciente_mais_novo['idade']} anos)")  
print(f"👴 Mais velho: {paciente_mais_velho['nome']} ({paciente_mais_velho['idade']} anos)")
```

*# Distribuição por faixa etária*

```
criancas = len([p for p in self.pacientes if p['idade'] < 18])  
adultos = len([p for p in self.pacientes if 18 <= p['idade'] < 60])  
idosos = len([p for p in self.pacientes if p['idade'] >= 60])
```

```
print(f"\n--- DISTRIBUIÇÃO POR IDADE ---")
print(f"Crianças (0-17): {criancas} ({(criancas/total)*100:.1f}%)")
print(f"Adultos (18-59): {adultos} ({(adultos/total)*100:.1f}%)")
print(f"Idosos (60+): {idosos} ({(idosos/total)*100:.1f}%)")
```

```
def buscar_paciente(self):
```

```
    """Busca paciente por nome"""
```

```
    if not self.pacientes:
```

```
        print("\n⚠ Nenhum paciente cadastrado ainda.")
```

```
        return
```

```
print("\n--- BUSCA DE PACIENTE ---")
```

```
termo_busca = input("Digite o nome (ou parte do nome): ").strip().lower()
```

```
if not termo_busca:
```

```
    print("ERRO: Digite um termo para busca.")
```

```
    return
```

```
# Busca por correspondência parcial
```

```
resultados = [p for p in self.pacientes
```

```
    if termo_busca in p['nome'].lower()]
```

```
if resultados:
```

```
    print(f"\n🔍 Encontrados {len(resultados)} resultado(s):")
```

```
    print("-" * 70)
```

```
    for paciente in resultados:
```

```
        print(f"ID: {paciente['id']:3d} | "
```

```
              f"Nome: {paciente['nome']:<25} | "
```

```
              f"Idade: {paciente['idade']:3d} | "
```

```
              f"Telefone: {paciente['telefone']}")
```

```
else:
```

```
    print(f"\n❌ Nenhum paciente encontrado com '{termo_busca}'")
```

```
def listar_pacientes(self):
```

```
    """Lista todos os pacientes cadastrados"""
```

```
    if not self.pacientes:
```

```
        print("\n⚠ Nenhum paciente cadastrado ainda.")
```

```
        return
```

```
print(f"\n--- LISTA DE PACIENTES ({len(self.pacientes)} total) ---")
```

```
print("-" * 85)
```

```
print(f"{'ID':<3} | {'NOME':<25} | {'IDADE':<5} | {'TELEFONE':<15} | {'CADASTRO'}")
```

```
print("-" * 85)
```

```
# Ordena por nome para melhor apresentação
```

```
pacientes_ordenados = sorted(self.pacientes, key=lambda p: p['nome'])
```

```
for paciente in pacientes_ordenados:
```

```
    print(f"{paciente['id']:<3} | "  
          f"{paciente['nome']:<25} | "  
          f"{paciente['idade']:<5} | "  
          f"{paciente['telefone']:<15} | "  
          f"{paciente['data_cadastro']}")
```

```
def exportar_dados(self):
```

```
    """Exporta dados dos pacientes para arquivo texto"""
```

```
    if not self.pacientes:
```

```
        print("\n⚠ Nenhum paciente cadastrado para exportar.")  
        return
```

```
    try:
```

```
        nome_arquivo = f"pacientes_clinica_{datetime.now().strftime('%Y%m%d_%H%M')}.txt"
```

```
        with open(nome_arquivo, 'w', encoding='utf-8') as arquivo:
```

```
            arquivo.write("=== CLÍNICA VIDA+ - RELATÓRIO DE PACIENTES ===\n")  
            arquivo.write(f>Data de geração: {datetime.now().strftime('%d/%m/%Y %H:%M:%S')}\n")  
            arquivo.write(f>Total de pacientes: {len(self.pacientes)}\n\n")
```

```
        for paciente in self.pacientes:
```

```
            arquivo.write(f>ID: {paciente['id']}\n")  
            arquivo.write(f>Nome: {paciente['nome']}\n")  
            arquivo.write(f>Idade: {paciente['idade']} anos\n")  
            arquivo.write(f>Telefone: {paciente['telefone']}\n")  
            arquivo.write(f>Data cadastro: {paciente['data_cadastro']}\n")  
            arquivo.write("-" * 40 + "\n")
```

```
        print(f"\n✓ Dados exportados com sucesso para: {nome_arquivo}")
```

```
    except Exception as e:
```

```
        print(f"\nERRO ao exportar dados: {e}")
```

```
def exibir_menu(self):
```

```
    """Exibe o menu principal"""
```

```
    print("\n" + "="*50)  
    print("    SISTEMA CLÍNICA VIDA+")  
    print("="*50)  
    print("1. 👤 Cadastrar paciente")  
    print("2. 📊 Ver estatísticas")  
    print("3. 🔍 Buscar paciente")  
    print("4. 📄 Listar todos os pacientes")  
    print("5. 💾 Exportar dados")  
    print("6. ❌ Sair")  
    print("="*50)
```



```

def executar(self):
    """Método principal do sistema"""
    print("🏥 Bem-vindo ao Sistema da Clínica Vida+!")
    print("Desenvolvido para otimizar o atendimento aos pacientes.")

    while True:
        try:
            self.exibir_menu()

            opcao = input("Escolha uma opção (1-6): ").strip()

            if opcao == '1':
                self.cadastrar_paciente()
            elif opcao == '2':
                self.calcular_estatisticas()
            elif opcao == '3':
                self.buscar_paciente()
            elif opcao == '4':
                self.listar_pacientes()
            elif opcao == '5':
                self.exportar_dados()
            elif opcao == '6':
                print("\n👋 Obrigado por usar o Sistema Clínica Vida+!")
                print("Sistema finalizado com sucesso.")
                break
            else:
                print(f"\n❌ Opção '{opcao}' inválida. Escolha entre 1 e 6.")

            input("\nPressione ENTER para continuar...")

        except KeyboardInterrupt:
            print("\n👋 Sistema encerrado pelo usuário.")
            break
        except Exception as e:
            print(f"\n❌ ERRO inesperado: {e}")
            input("Pressione ENTER para continuar...")






# Inicialização do sistema
if __name__ == "__main__":
    sistema = SistemaClinicaVidaMais()
    sistema.executar()

```

## 2.3 Exemplo de Execução

=== SISTEMA CLÍNICA VIDA+ ===

1. 🧑 Cadastrar paciente

2.  Ver estatísticas
3.  Buscar paciente
4.  Listar todos os pacientes
5.  Exportar dados
6.  Sair

Escolha uma opção (1-6): 1

--- CADASTRO DE PACIENTE ---

Nome do paciente: João Silva

Idade: 45

Telefone (XX) XXXXX-XXXX: (11) 99999-9999

✓ Paciente João Silva cadastrado com sucesso!

## 2.4 Recursos Implementados

### Funcionalidades Principais:

- Cadastro com validação rigorosa de dados
- Cálculos estatísticos automáticos
- Busca por correspondência parcial
- Listagem organizada e ordenada
- Exportação de dados para arquivo

### Tratamento de Erros:

- Validação de formato de telefone
- Verificação de idades válidas
- Proteção contra entradas vazias
- Interceptação de interrupções do usuário

---

## PASSO 3: ANÁLISE DE SISTEMAS LÓGICOS

### 3.1 Definição das Variáveis Lógicas

#### Variáveis do Sistema:

- **A:** Paciente tem agendamento marcado
- **B:** Paciente está com documentos em dia (RG/CPF válidos)
- **C:** Há médico disponível no horário
- **D:** Paciente está em dia com pagamentos anteriores

### 3.2 Expressões Lógicas

Para Consulta Normal:

(A ∧ B ∧ C) ∨ (B ∧ C ∧ D)

Para Emergência:

C ∧ (B ∨ D)

### 3.3 Tabela Verdade - Consulta Normal

A	B	C	D	A ∧ B ∧ C	B ∧ C ∧ D	(A ∧ B ∧ C) ∨ (B ∧ C ∧ D)
F	F	F	F	F	F	F
F	F	F	V	F	F	F
F	F	V	F	F	F	F
F	F	V	V	F	F	F
F	V	F	F	F	F	F
F	V	F	V	F	F	F
F	V	V	F	F	F	F
F	V	V	V	F	V	V
V	F	F	F	F	F	F
V	F	F	V	F	F	F
V	F	V	F	F	F	F
V	F	V	V	F	F	F
V	V	F	F	F	F	F
V	V	F	V	F	F	F
V	V	V	F	V	F	V
V	V	V	V	V	V	V

### 3.4 Tabela Verdade - Emergência

A	B	C	D	B ∨ D	C ∧ (B ∨ D)
F	F	F	F	F	F
F	F	F	V	V	F
F	F	V	F	F	F
F	F	V	V	V	V
F	V	F	F	V	F
F	V	F	V	V	F

A	B	C	D	$B \vee D$	$C \wedge (B \vee D)$
F	V	V	F	V	V
F	V	V	V	V	V
V	F	F	F	F	F
V	F	F	V	V	F
V	F	V	F	F	F
V	F	V	V	V	V
V	V	F	F	V	F
V	V	F	V	V	F
V	V	V	F	V	V
V	V	V	V	V	V

### 3.5 Análise dos Resultados

#### Consulta Normal:

- Situações de atendimento: 3 de 16 (18,75%)
- Condições mais restritivas devido à necessidade de documentação

#### Emergência:

- Situações de atendimento: 6 de 16 (37,5%)
- Maior flexibilidade para casos urgentes

### 3.6 Situação Prática

#### Condições do paciente:

- Sem agendamento ( $A = F$ )
- Documentos OK ( $B = V$ )
- Médico disponível ( $C = V$ )
- Pagamentos atrasados ( $D = F$ )

**Avaliação Consulta Normal:**  $(F \wedge V \wedge V) \vee (V \wedge V \wedge F) = F \vee F = \mathbf{F}$  (NÃO será atendido)

**Avaliação Emergência:**  $V \wedge (V \vee F) = V \wedge V = \mathbf{V}$  (SERÁ atendido)

**Conclusão:** O paciente não pode ser atendido em consulta normal, mas pode ser atendido em caso de emergência.

## PASSO 4: ALGORITMO DE FILA DE ATENDIMENTO

### 4.1 Análise da Estrutura de Dados

A fila (queue) é uma estrutura de dados que segue o princípio FIFO (First In, First Out), sendo ideal para organizar atendimentos médicos onde a ordem de chegada deve ser respeitada.

### 4.2 Pseudocódigo do Algoritmo

```
ALGORITMO FilaAtendimentoClinica
```

```
VAR
```

```
    fila_pacientes: FILA DE REGISTRO
```

```
    paciente: REGISTRO
```

```
        nome: CADEIA
```

```
        cpf: CADEIA
```

```
FIM_REGISTRO
```

```
contador: INTEIRO
```

```
opcao: INTEIRO
```

```
INÍCIO
```

```
    contador ← 0
```

```
    ESCRIVA "=== SISTEMA DE FILA DE ATENDIMENTO CLÍNICA VIDA+ ==="
```

```
    // Inserir 3 pacientes na fila
```

```
    PARA contador DE 1 ATÉ 3 FAÇA
```

```
        ESCRIVA "--- CADASTRO DO PACIENTE", contador, "---"
```

```
        ESCRIVA "Nome do paciente:"
```

```
        LEIA paciente.nome
```

```
        ENQUANTO VAZIO(paciente.nome) FAÇA
```

```
            ESCRIVA "ERRO: Nome não pode estar vazio!"
```

```
            LEIA paciente.nome
```

```
        FIM_ENQUANTO
```

```
        ESCRIVA "CPF (XXX.XXX.XXX-XX):"
```

```
        LEIA paciente.cpf
```

```
        ENQUANTO NÃO VALIDAR_CPF(paciente.cpf) FAÇA
```

```
            ESCRIVA "ERRO: CPF inválido! Use o formato XXX.XXX.XXX-XX"
```

```
            LEIA paciente.cpf
```

```
        FIM_ENQUANTO
```

```
        INSERIR_FILA(fila_pacientes, paciente)
```

```
        ESCRIVA "Paciente", paciente.nome, "adicionado à fila."
```

```
FIM_PARA
```

ESCREVA ""

ESCREVA "=== FILA INICIAL ==="

MOSTRAR\_FILA(fila\_pacientes)

// Remover primeiro paciente para atendimento

SE NÃO VAZIA(fila\_pacientes) ENTÃO

paciente ← REMOVER\_FILA(fila\_pacientes)

ESCREVA ""

ESCREVA ">>> CHAMANDO PARA ATENDIMENTO <<<"

ESCREVA "Paciente:", paciente.nome

ESCREVA "CPF:", paciente.cpf

ESCREVA "Direcionado para o consultório."

FIM\_SE

// Mostrar fila após primeiro atendimento

ESCREVA ""

ESCREVA "=== FILA APÓS PRIMEIRO ATENDIMENTO ==="

SE VAZIA(fila\_pacientes) ENTÃO

ESCREVA "A fila está vazia."

SENÃO

MOSTRAR\_FILA(fila\_pacientes)

FIM\_SE

FIM

FUNÇÃO VALIDAR\_CPF(cpf: CADEIA): LÓGICO

INÍCIO

// Validação básica de formato XXX.XXX.XXX-XX

SE TAMANHO(cpf) = 14 E

cpf[4] = '.' E cpf[8] = '.' E cpf[12] = '-' ENTÃO

RETORNA VERDADEIRO

SENÃO

RETORNA FALSO

FIM\_SE

FIM\_FUNÇÃO

PROCEDIMENTO MOSTRAR\_FILA(fila: FILA DE REGISTRO)

VAR

temp\_fila: FILA DE REGISTRO

paciente: REGISTRO

posicao: INTEIRO

INÍCIO

posicao ← 1

temp\_fila ← COPIAR\_FILA(fila)

ESCREVA "Posição | Nome do Paciente | CPF"

```
ESCREVA "-----|-----|-----"
```

```
ENQUANTO NÃO VAZIA(temp_fila) FAÇA
```

```
    paciente ← REMOVER_FILA(temp_fila)
```

```
    ESCRIBA posicao, "°   |", paciente.nome, "|", paciente.cpf
```

```
    posicao ← posicao + 1
```

```
FIM_ENQUANTO
```

```
    ESCRIBA "Total de pacientes na fila:", TAMANHO(fila)
```

```
FIM_PROCEDIMENTO
```

```
PROCEDIMENTO INSERIR_FILA(fila: FILA DE REGISTRO, item: REGISTRO)
```

```
INÍCIO
```

```
    ADICIONAR_NO_FIM(fila, item)
```

```
FIM_PROCEDIMENTO
```

```
FUNÇÃO REMOVER_FILA(fila: FILA DE REGISTRO): REGISTRO
```

```
INÍCIO
```

```
    RETORNA REMOVER_DO_INICIO(fila)
```

```
FIM_FUNÇÃO
```

```
FUNÇÃO VAZIA(fila: FILA DE REGISTRO): LÓGICO
```

```
INÍCIO
```

```
    RETORNA TAMANHO(fila) = 0
```

```
FIM_FUNÇÃO
```

## 4.3 Exemplo de Execução

```
=== SISTEMA DE FILA DE ATENDIMENTO CLÍNICA VIDA+ ===
```

```
--- CADASTRO DO PACIENTE 1 ---
```

```
Nome do paciente: Maria Santos
```

```
CPF (XXX.XXX.XXX-XX): 123.456.789-01
```

```
Paciente Maria Santos adicionado à fila.
```

```
--- CADASTRO DO PACIENTE 2 ---
```

```
Nome do paciente: José Silva
```

```
CPF (XXX.XXX.XXX-XX): 987.654.321-00
```

```
Paciente José Silva adicionado à fila.
```

```
--- CADASTRO DO PACIENTE 3 ---
```

```
Nome do paciente: Ana Costa
```

```
CPF (XXX.XXX.XXX-XX): 456.789.123-45
```

```
Paciente Ana Costa adicionado à fila.
```

```
=== FILA INICIAL ===
```

Posição | Nome do Paciente | CPF

-----|-----|-----

1° | Maria Santos | 123.456.789-01

2° | José Silva | 987.654.321-00

3° | Ana Costa | 456.789.123-45

Total de pacientes na fila: 3

>>> CHAMANDO PARA ATENDIMENTO <<<

Paciente: Maria Santos

CPF: 123.456.789-01

Direcionado para o consultório.

=== FILA APÓS PRIMEIRO ATENDIMENTO ===

Posição | Nome do Paciente | CPF

-----|-----|-----

1° | José Silva | 987.654.321-00

2° | Ana Costa | 456.789.123-45

Total de pacientes na fila: 2

## 4.4 Análise da Complexidade

### Complexidade Temporal:

- Inserção:  $O(1)$  - constante
- Remoção:  $O(1)$  - constante
- Visualização:  $O(n)$  - linear

**Complexidade Espacial:**  $O(n)$  onde  $n$  é o número de pacientes na fila

---

## PASSO 5: DIAGRAMA DE CASOS DE USO

### 5.1 Identificação dos Atores

#### Atores Primários:

- **Secretária:** Responsável por cadastros e agendamentos
- **Médico:** Realiza atendimentos e prescrições
- **Paciente:** Recebe atendimento médico

#### Atores Secundários:

- **Sistema de Impressão:** Imprime receitas automaticamente

### 5.2 Identificação dos Casos de Uso

#### Casos de Uso Principais:



1. **Cadastrar Paciente**
2. **Agendar Consulta**
3. **Confirmar Consulta**
4. **Cancelar Consulta**
5. **Gerar Receita**
6. **Imprimir Receita**

## 5.3 Especificação Detalhada dos Casos de Uso

### UC001 - Cadastrar Paciente

- **Ator Principal:** Secretária
- **Pré-condição:** Sistema disponível
- **Fluxo Principal:**
  1. Secretária acessa módulo de cadastro
  2. Sistema apresenta formulário de cadastro
  3. Secretária preenche dados do paciente
  4. Sistema valida informações
  5. Sistema confirma cadastro realizado
- **Pós-condição:** Paciente cadastrado no sistema

### UC002 - Agendar Consulta

- **Ator Principal:** Secretária
- **Pré-condição:** Paciente deve estar cadastrado
- **Fluxo Principal:**
  1. Secretária seleciona opção agendar consulta
  2. Sistema solicita identificação do paciente
  3. Secretária informa dados do paciente
  4. Sistema verifica se paciente está cadastrado
  5. Sistema apresenta horários disponíveis
  6. Secretária seleciona horário desejado
  7. Sistema confirma agendamento
- **Fluxo Alternativo:** Paciente não cadastrado
  - 4a. Sistema informa que paciente não está cadastrado
  - 4b. Include UC001 - Cadastrar Paciente

### UC003 - Confirmar Consulta

- **Ator Principal:** Secretária
- **Pré-condição:** Consulta deve estar agendada
- **Fluxo Principal:**
  1. Secretária acessa lista de consultas agendadas
  2. Sistema apresenta consultas do dia
  3. Secretária seleciona consulta para confirmar
  4. Sistema atualiza status para confirmado
- **Pós-condição:** Consulta marcada como confirmada

### UC004 - Cancelar Consulta

- **Ator Principal:** Médico ou Secretária
- **Pré-condição:** Consulta deve estar agendada
- **Fluxo Principal:**
  1. Ator acessa sistema de consultas
  2. Sistema apresenta lista de consultas
  3. Ator seleciona consulta para cancelar
  4. Sistema solicita motivo do cancelamento
  5. Ator informa motivo
  6. Sistema confirma cancelamento
- **Pós-condição:** Consulta cancelada e horário liberado

### UC005 - Gerar Receita

- **Ator Principal:** Médico
- **Pré-condição:** Paciente deve estar em atendimento
- **Fluxo Principal:**
  1. Médico acessa módulo de prescrição
  2. Sistema apresenta dados do paciente
  3. Médico preenche prescrição médica
  4. Sistema valida dados da receita
  5. Sistema salva receita no histórico
  6. Include UC006 - Imprimir Receita
- **Pós-condição:** Receita gerada e associada ao paciente

### UC006 - Imprimir Receita

- **Ator Principal:** Sistema de Impressão
- **Pré-condição:** Receita deve estar gerada
- **Fluxo Principal:**
  1. Sistema aciona impressora
  2. Sistema formata receita para impressão
  3. Sistema envia dados para impressora
  4. Sistema confirma impressão realizada
- **Pós-condição:** Receita impressa fisicamente

## 5.4 Relacionamentos Entre Casos de Uso

### Relacionamentos Identificados:

#### 1. Include:

- UC002 (Agendar Consulta) include UC001 (Cadastrar Paciente)
- UC005 (Gerar Receita) include UC006 (Imprimir Receita)

#### 2. Extend:

- UC001 (Cadastrar Paciente) pode ser estendido durante UC002 (Agendar Consulta)

## 5.5 Representação Textual do Diagrama

Sistema de Gestão da Clínica Vida+

Atores:

- Secretária
- Médico
- Sistema de Impressão

Casos de Uso e Relacionamentos:

[Secretária] —→ (Cadastrar Paciente)

[Secretária] —→ (Agendar Consulta) ←—include—→ (Cadastrar Paciente)

[Secretária] —→ (Confirmar Consulta)

[Médico] —→ (Cancelar Consulta)

[Secretária] —→ (Cancelar Consulta)

[Médico] —→ (Gerar Receita) ←—include—→ (Imprimir Receita) ←— [Sistema de Impressão]

Condições de Pré-requisito:

- Para agendar consulta: paciente deve estar cadastrado
- Para confirmar consulta: consulta deve estar agendada

- Para cancelar consulta: consulta deve existir
- Para gerar receita: paciente deve estar em atendimento

## 5.6 Regras de Negócio

**RN001:** Um paciente só pode ser agendado se estiver previamente cadastrado no sistema.

**RN002:** Médicos e secretárias podem cancelar consultas, mas apenas médicos podem gerar receitas.

**RN003:** Sempre que uma receita for gerada, o sistema deve automaticamente acionar a impressão.

**RN004:** O sistema deve manter histórico completo de todas as operações realizadas.

**RN005:** Consultas confirmadas têm prioridade sobre consultas apenas agendadas.

---

## CONCLUSÃO

O desenvolvimento da plataforma de saúde para a Clínica Vida+ demonstrou a aplicação prática de conceitos fundamentais da Análise e Desenvolvimento de Sistemas. Através dos cinco passos realizados, foi possível abordar aspectos críticos do desenvolvimento de software: metodologia ágil, programação orientada a problemas reais, análise lógica de sistemas, estruturas de dados eficientes e modelagem de requisitos.

A implementação da metodologia Scrum através do Trello proporcionou uma visão clara do processo de desenvolvimento iterativo e incremental, evidenciando como a organização e o planejamento são essenciais para o sucesso de projetos de tecnologia. A divisão em sprints permitiu foco nas funcionalidades prioritárias e entrega contínua de valor.

O sistema de cadastro desenvolvido em Python demonstrou a importância de validações robustas e tratamento adequado de erros, aspectos fundamentais para sistemas que lidam com dados sensíveis como informações médicas. As funcionalidades implementadas atendem diretamente às necessidades identificadas na Clínica Vida+, proporcionando eficiência operacional.

A análise dos sistemas lógicos revelou como regras de negócio complexas podem ser modeladas através de álgebra booleana, permitindo automatização de processos decisórios críticos como controle de acesso a atendimentos. A diferenciação entre consultas normais e emergenciais demonstra a flexibilidade necessária em sistemas de saúde.

O algoritmo de fila implementado mostra como estruturas de dados apropriadas podem otimizar processos organizacionais, respeitando princípios de equidade no atendimento. A implementação FIFO garante que a ordem de chegada seja respeitada, princípio fundamental em serviços de saúde.

O diagrama de casos de uso consolidou a visão sistêmica do projeto, identificando atores, funcionalidades e relacionamentos que direcionarão o desenvolvimento futuro da plataforma. A

modelagem clara de requisitos é essencial para alinhamento entre stakeholders e equipe de desenvolvimento.

Este projeto integrado evidenciou que o desenvolvimento de sistemas de saúde requer não apenas competências técnicas, mas também compreensão profunda dos processos de negócio e necessidades dos usuários finais. A solução proposta oferece base sólida para modernização dos processos da Clínica Vida+, com potencial de expansão para funcionalidades mais avançadas.

A experiência adquirida neste projeto demonstra a importância da abordagem multidisciplinar na formação do analista e desenvolvedor de sistemas, preparando profissionais capazes de compreender, analisar e propor soluções tecnológicas eficazes para problemas complexos do mundo real.

---

## REFERÊNCIAS

PRESSMAN, Roger S.; MAXIM, Bruce R. **Engenharia de Software: uma abordagem profissional**. 8. ed. Porto Alegre: AMGH, 2016.

SOMMERVILLE, Ian. **Engenharia de Software**. 10. ed. São Paulo: Pearson Education do Brasil, 2018.

SCHWABER, Ken; SUTHERLAND, Jeff. **O Guia do Scrum**. Scrum.org, 2020. Disponível em: <https://scrumguides.org/>. Acesso em: 03 set. 2025.

FORBELLONE, André Luiz Villar; EBERSPÄCHER, Henri Frederico. **Lógica de Programação: a construção de algoritmos e estruturas de dados**. 3. ed. São Paulo: Prentice Hall, 2005.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML: Guia do Usuário**. 2. ed. Rio de Janeiro: Elsevier, 2005.

FOWLER, Martin. **UML Essencial: um breve guia para linguagem padrão de modelagem de objetos**. 3. ed. Porto Alegre: Bookman, 2005.

MENEZES, Paulo Fernando Blauth. **Matemática Discreta para Computação e Informática**. 4. ed. Porto Alegre: Bookman, 2013.

CORMEN, Thomas H. et al. **Algoritmos: teoria e prática**. 3. ed. Rio de Janeiro: Elsevier, 2012.

MINISTÉRIO DA SAÚDE. **Manual de Gestão da Informação do SUS**. Brasília: Ministério da Saúde, 2023.

CONSELHO FEDERAL DE MEDICINA. **Resolução CFM nº 1.821/2007**. Aprova as normas técnicas concernentes à digitalização e uso dos sistemas informatizados para a guarda e manuseio dos documentos dos prontuários dos pacientes. Brasília, 2007.