UNIVERSIDADE FEDERAL DE MINAS GERAIS

Programa de Graduação em Sistemas de Informação – PDS II

Abner D. Bicalho de Lima

Cybele F. Cançado

João Pedro C. Cyrino

Marçal Augusto Morais

Samuel William Almeida Santos

IMPLEMENTAÇÃO DE UM PROGRAMA DE GERENCIAMENTO DE AGENDA EM UMA CLÍNICA PSICOLÓGICA

Belo Horizonte

2019

Abner D. Bicalho de Lima

Cybele F. Cançado

João Pedro C. Cyrino

Marçal Augusto Morais

Samuel William Almeida Santos

IMPLEMENTAÇÃO DE UM PROGRAMA DE GERENCIAMENTO DE AGENDA EM UMA CLÍNICA PSICOLÓGICA

Trabalho apresentado ao programa de Graduação em Sistemas de Informação na Universidade Federal de Minas Gerais, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Orientador: Prof. Júlio César Soares dos Reis

Área de concentração: Programação e Estrutura de Dados II

Belo Horizonte

2019

SUMÁRIO

1 INTRODUÇÃO

A utilização da linguagem de programação de computadores traz consigo uma vasta gama de recursos que possibilitam a implementação desde simples programas para cálculos matemáticos até a construção de sistemas de engenharia complexos. Isso tem tornado mais eficaz e eficiente os processos de gerenciamento de dados, uma vez que as ferramentas computacionais passam a assumir atividades rotineiras e geram resultados precisos para a tomada de decisão do usuário eliminando assim, por exemplo, tempo demasiado de processamento, arquivos complexos (desburocratiza o processo) e acumulo de erros ao longo de uma cadeia produtiva.

Dado a gama de benefícios observados no estudo da implementação de sistemas computacionais, realizou-se uma análise em uma clínica psicológica social Acolhida, cuja a principal demanda de controle e gerenciamento é dada por arquivos físicos e manuais, e sobre isto, foi desenvolvido um programa de sistema de gestão computacional *(Enterprise Resource Planning ERP)*, na linguagem C++. A implantação engloba a aplicação dos conceitos de sistemas sobre o controle de gestão de agenda, geração de relatórios, cadastro de funcionários e clientes, permitindo adicionar ou consultar informações além da geração de uma agenda que cruze as características dos pacientes e psicólogos com suas respectivas disponibilidades e agendamento de consultas.

A Acolhida é uma clínica social de psicologia, vinculada à Igreja Nossa Senhora da Divina Providência, localizada no bairro Ouro Preto em Belo Horizonte. Os trabalhos na clínica iniciaram no ano de 1995, com três pacientes, dois psicólogos e duas secretarias. A clínica funciona desde então atendendo a demanda da população de baixa renda da região com profissionais, psicólogos e secretarias, voluntários. A clínica, para os pacientes que podem contribuir, cobra um valor que varia de pessoa para pessoa, sendo trinta reais o valor máximo cobrado por mês para cobrir as despesas de luz, água e limpeza da casa. Atualmente, trabalham voluntariamente na Acolhida, vinte e cinco psicólogos e doze secretarias que atendem 82 pacientes de segunda a sexta feira de sete às dezoito horas. Desde o início dos atendimentos, 2551 pessoas já foram atendidas na Acolhida, pessoas estas que procuram espontaneamente ou são encaminhadas pela

creche e escola municipal que estão instaladas na região, bem como, em alguns casos são encaminhadas pelo Posto de Saúde.

O trabalho apresenta conceitos da disciplina de Programação e Desenvolvimento de Software II, do semestre 2019/2 da Universidade Federal de Minas Gerais. A estruturação do programa utiliza os recursos de Programação Orientada a Objeto (POO), aplicando os conceitos de classe, herança, polimorfismo, encapsulamento, modularização, testes, dentre outros. O projeto pode ser encontrado em https://github.com/pds2/20192-team-9.

2 DETALHAMENTO DO PROJETO

A primeira análise realizada trata-se do entendimento do fluxo de informação que é gerada ao longo do processo de atendimento do cliente e as ações de gestão pro de trás do fluxo gerencial. O processo se inicia com o cadastro dos funcionários que se dividem em dois, sendo eles secretários e psicólogos. Nesta etapa, o psicólogo cadastra sua disponibilidade de atendimento, enquanto o a secretária se torna a responsável pelo cadastro do banco de dados dos clientes. Na medida que surge o horário disponível, cria-se o processo de agendamento da consulta, combinando os horários disponíveis do psicólogo e do cliente. Após isto, o atendimento é firmado e após a consulta, é gerado o prontuário. Caso o psicólogo receite alta, ou o cliente deseje interromper o tratamento, o psicólogo gera então um prontuário final e os dados do cliente passam a fazer parte da base de banco de dados, visto que o mesmo pode retornar o tratamento em datas posteriores. O fluxograma na fig. 1 demonstra com clareza os processos citados.

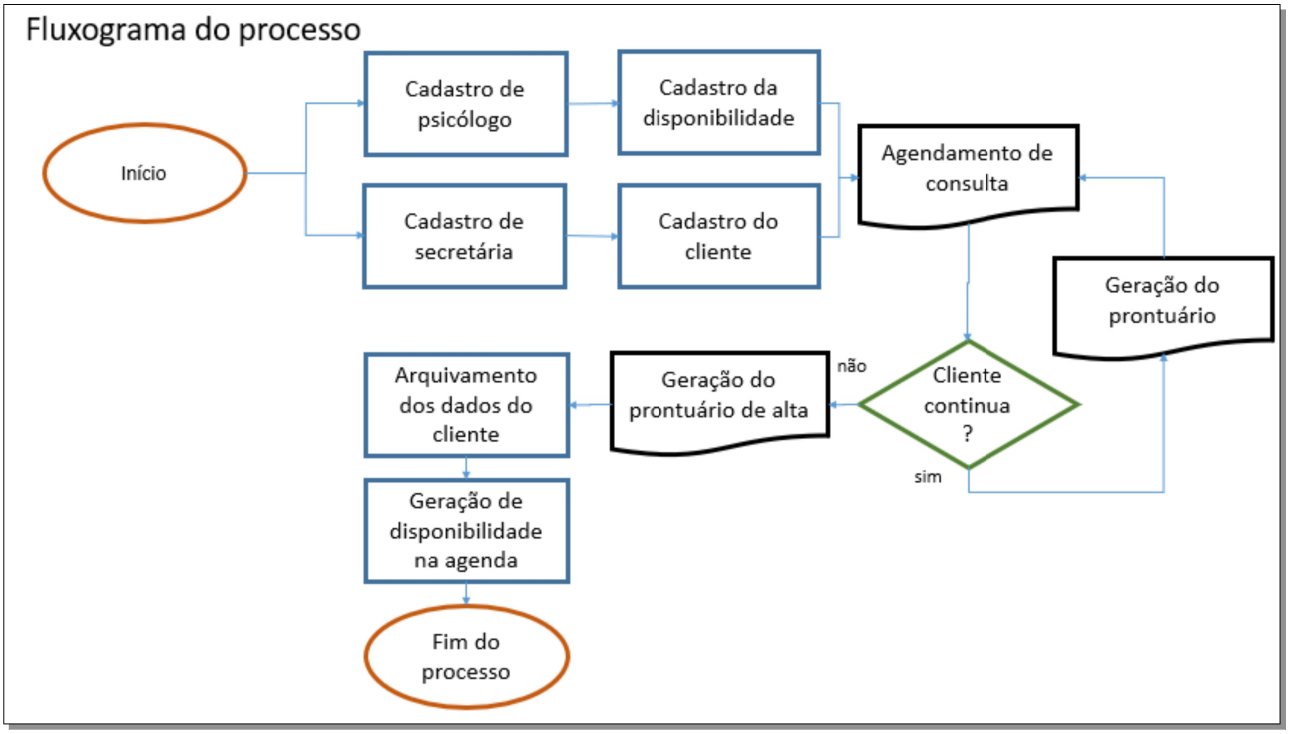

Fig.1 – Fluxograma do processo de agendamento e consulta da clínica.

Com base neste estudo, inicia-se então o processo de montagem das principais classes que serão implantadas, cada qual com sua respectiva função.

3 IMPLEMENTAÇÃO DO PROGRAMA

Para a implementação do programa, uma série de movimentos devem ser observados, sendo elas as definições das classes, métodos de encapsulamento, definição das funções, herança, polimorfismo etc. A definição dos conceitos nos permite elaborar programas mais bem estruturados e com os métodos de modularização, torna o acesso facilitado para gerenciamento de futuros updates do sistema ou até mesmo revisão do código implantado. Vale observar que as funções implementadas também são de extremo apoio para sua identificação e evitar o retrabalho de redundância do código.

3.1 Definição de classes

Para a organização do sistema, foi criado diferentes classes que encapsulam os dados e métodos como forma de definir os poderes do usuário sobre o sistema, como também privá-los de informações que não condiz com a classe. Dessa forma, o agrupamento se deu da seguinte maneira:

Pessoa (Super-Classe): É uma classe que possui atributos como nome, cpf e endereço e métodos que serão a base para suas classes derivadas.

Psicólogo | *Paciente (Sub-Classes):* São 2 classes que herdam informações de Pessoa. Estas possuem a função de modularizar e encapsular os diferentes tipos de objetos que serão utilizados no programa, armazenando informações e métodos pertinentes á elas próprias. São usadas para relacionar informações entre si e entre classes que veremos a seguir.

Agenda: Esta não possui herança. Tem como função o controle de consultas da clínica, por meio de armazenamento de datas e horários, funções que serão utilizadas por outras classes. Ela estabelece uma relação de composição com Psicólogo, pois é criado um objeto do tipo Agenda dentro de psicólogo.

Sistema: Por fim, essa classe é a que gerencia todo o programa. Tem como objetivo prover métodos que coordenam e associam as classes para prover funcionalidade ao programa. Ela faz a conversa entre classes diferentes, criando ambientes, facilitadores e operações, que cumprem a funcionalidade do sistema.

O diagrama 1 demonstra as inter-relações do exposto acima, referente as classes, subclasses e processo de herança.

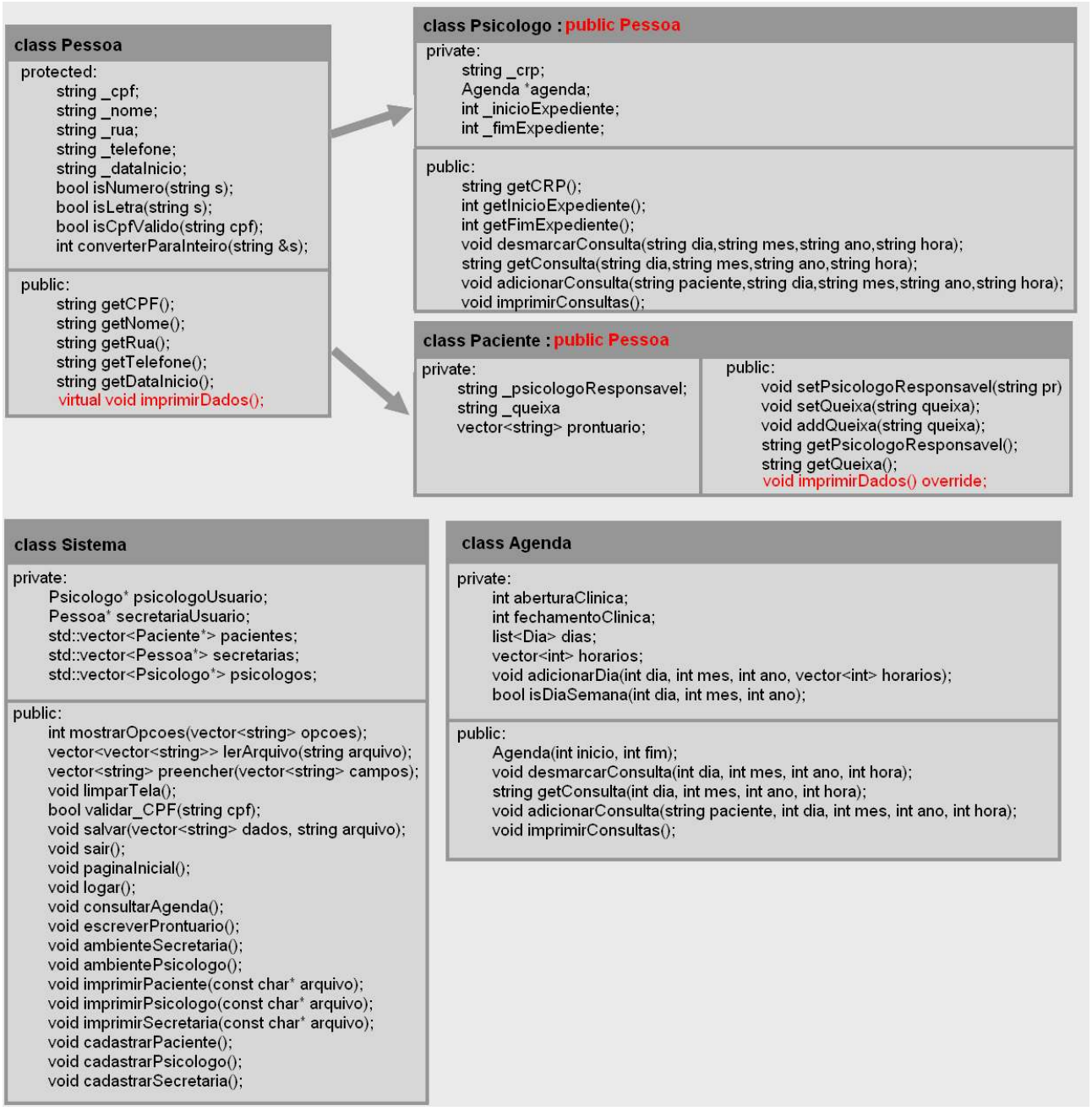

Diagrama 1: Demonstração das classes com suas respectivas heranças, atributos e métodos.

Dessa forma, o programa é inicializado no main com a chamada da função paginaInicial() da classe sistema, e todas as outras funcionalidades são interligadas por essa função, de acordo com o acesso e cadastro dado aos usuários do sistema.

3.2 Análise de Complexidade

Nesta secção, iremos analisar os métodos de cada classe e fazer um resumo de suas funcionalidades.

3.2.1 Classe Pessoa:

std::string getCPF(): Getter para consultar o atributo CPF da classe Pessoa, pois é protegido.

std::string getRua(): Getter para consultar o atributo Rua da classe Pessoa, pois é protegido.

std::string getNome(): Getter para consultar o atributo Nome da classe Pessoa, pois é protegido.

std::string getTelefone(): Getter para consultar o atributo Telefone da classe Pessoa, pois é protegido.

std::string getDataInicio(): Getter para consultar o atributo DataInicio da classe Pessoa, pois é protegido.

bool isNumero(std::string s): Checa se a string é composta completamente por números.

bool isLetra(std::string s): Checa se a string é composta completamente por letras e espaços.

bool isCpfValido(string cpf): Checa se a string passada como parâmetro é um CPF válido (Possui 11 números).

int converterParaInteiro(std::string &s): Verifica se a entrada recebida é um número. Se não for, faz tratamento de exceção e pede ao usuário para digitar um número.

virtual void imprimirDados(): Função que faz o uso de polimorfismo. Objetivo principal é imprimir dados da classe em que está implementada.

3.2.2 Classe Psicólogo:

std::string getCRP(): Getter para consultar o atributo CRP da classe Psicólogo, pois é protegido.

int getInicioExpediente(): Getter para consultar o atributo _inicioExpediente da classe Psicólogo, pois é protegido.

int getFimExpediente(): Getter para consultar o atributo _fimExpediente da classe Psicólogo, pois é protegido.

void desmarcarConsulta(std::string in_dia, std::string in_mes, std::string in_ano, std::string in_hora): Função que permite o psicólogo desmarcar uma consulta por meio da classe Agenda. Deve ser acionada passando parametros da data e horário para selecionar a consulta a ser cancelada.

std::string getConsulta(std::string in_dia, std::string in_mes, std::string in_ano, std::string in_hora): Função que permite o psicólogo visualizar uma consulta por meio da classe Agenda. Deve ser acionada passando parametros da data e horário para selecionar a consulta a ser visualizada.

void adicionarConsulta(std::string in_paciente, std::string in_dia, std::string in_mes, std::string in_ano, std::string in_hora): Função que permite o psicólogo marcar uma consulta por meio da classe Agenda. Deve ser acionada passando parametros da data e horário para selecionar a consulta a ser marcada.

void imprimirConsultas(): Função que acessa a função da classe Agenda para imprimir todas as próximas consultas marcadas.

3.2.3 Classe Paciente:

void setPsicologoResponsavel(std::string pr): Setter para setar o atributo PsicologoResponsavel da classe Paciente, pois é protegido.

void setQueixa(std::string queixa): Setter para setar o atributo Queixa da classe Paciente, pois é protegido.

void addQueixa(std::string queixa): Função que concatena uma nova queixa à uma anterior da classe Paciente.

std::string getPsicologoResponsavel(): Getter para consultar o atributo PsicologoResponsavel da classe Paciente, pois é protegido.

std::string getQueixa(): Getter para consultar o atributo Queixa da classe Paciente, pois é protegido.

void imprimirDados() override: Função override que modifica a função da Super-Classe Pessoa para que possa imprimir as informações da classe, diferenciando pelos atributos exclusivos.

3.2.4 Classe Agenda:

void adicionarDia(int dia, int mes, int ano, std::vector<int> horarios): Função que seta os dias disponíveis e permitidos na agenda.

bool isDiaSemana(int dia, int mes, int ano): Função que verifica se o dia passado como parâmetro é dia da semana.

void desmarcarConsulta(int dia, int mes, int ano, int hora): Função que desmarca uma consulta na agenda, por meio de parametros de data e hora.

std::string getConsulta(int dia, int mes, int ano, int hora): Função que visualiza uma consulta na agenda, por meio de parametros de data e hora.

void adicionarConsulta(std::string paciente, int dia, int mes, int ano, int hora): Função que marca uma consulta na agenda, por meio de parametros de data e hora.

void imprimirConsultas(): Função que imprime todas as próximas consultas da agenda.

3.2.5 Classe Sistema:

int mostrarOpcoes(std::vector<std::string> opções): Função que mostra um menu de opções da tela inicial para o usuário do programa.

std::vector<std::vector<std::string>> lerArquivo(std::string arquivo): Função que lê um arquivo para coletar informações salvas.

std::vector<std::string> preencher(std::vector<std::string> campos): Função que preenche um CPF e chama outra função para verificar se ele é válido.

void limparTela(): Função que limpa a tela do programa para entrar em outra tela.

bool validar_CPF(std::string cpf): Função que valida/invalida um CPF passado como parâmetro.

void salvar(std::vector<std::string> dados, std::string arquivo): Função que salva dados preenchidos no programa em um arquivo. Recebe os dados a serem gravados e o arquivo a ser preenchido.

void sair(): Função que finaliza o programa.

void paginaInicial(): Função que mostra as opções da tela inicial.

void logar(): Função que recebe um CPF e chama outras funções para a verificação do CPF e validação de usuário no login.

void consultarAgenda(): Função que chama a função da Agenda para imprimir consultas.

void escreverProntuario(): Função que escreve um prontuário para a classe Paciente.

void ambienteSecretaria(): Função que cria um ambiente após o usuário logar como Secretária, onde é mostrado as opções disponíveis para tal usuario.

void ambientePsicologo(): Função que cria um ambiente após o usuário logar como Psicólogo, onde é mostrado as opções disponíveis para tal usuario.

void imprimirPaciente(const char* arquivo): Função que, a partir do Arquivo de pacientes salvo, mostra todos os cadastros dos mesmos.

void imprimirPsicologo(const char* arquivo): Função que, a partir do Arquivo de psicólogos salvo, mostra todos os cadastros dos mesmos.

void imprimirSecretaria(const char* arquivo): Função que, a partir do Arquivo de secretárias salvo, mostra todos os cadastros dos mesmos.

void cadastrarPaciente(): Função que coleta as informações de um paciente a ser cadastrado e as salva em um Arquivo de pacientes.

void cadastrarPsicologo(): Função que coleta as informações de um psicólogo a ser cadastrado e as salva em um Arquivo de psicólogos.

void cadastrarSecretaria(): Função que coleta as informações de uma secretária a ser cadastrada e as salva em um Arquivo de secretária.

4 TESTES DE UNIDADE

Foram criados arquivos de testes exclusivos para cada classe, para verificação se, mediante determinadas entradas, o programa retorna valores esperados para algumas funções, conforme as descritas abaixo:

testesAgenda.cpp

TEST_CASE("01 - Testando construtor Agenda"): Testa se as entradas da agenda só aceitam entrada no horário comercial.

TEST_CASE("02 - Marcando e Desmarcando Consultas"): verifica se ao desmarcar a consulta, o espaço na agenda é liberado para outro cliente.

TEST_CASE("02 - Marcando e Desmarcando Consultas - Excecoes"): verifica se é possível marcar um horário intercalado a outro; se é possível agendar horários fora do horário comercial; se é possível agendar horários durante o fim de semana.

testesPaciente.cpp

TEST_CASE("01 - Testando Construtor, setters e getters"): Checa se os dados do paciente estão sendo armazenados no banco de dados.

testePessoa.cpp

TEST_CASE("01 - Testando Construtor e getters"): checa se os dados de nome não possuem números; se o cpf não aceita dados do tipo números com menos ou mais de 11 números, e também se não aceita letras; Se o telefone não aceita letras; se a data é composta apenas de letras.

testePsicologo.cpp

TEST_CASE("01 - Testando construtor e getters"): checa se os dados de nome não possuem números; se o cpf não aceita dados do tipo números com menos ou mais de 11 números, e também se não aceita letras; Se o telefone não aceita letras; se a data é composta apenas de letras; se a data aceita apenas números; se o CRP é composto apenas de números; se o horário de entrada e saída são compostos apenas de um número.

TEST_CASE("02 - Testando operacoes da agenda"): verifica se a visualização da agenda encontra-se disponível para o psicólogo.

testeSistema.cpp

TEST_CASE("01 - Testando Construtor"): Checa se a entrada através do cpf irá direcionar o usuário a página correta. Caso ele seja um psicólogo, ele deve ser direcionado ao ambiente de trabalho do psicólogo, idem secretária.

5 POLIMORFISMO

Podemos encontrar conceitos de polimorfismo nas classes Pessoa e Paciente, onde a primeira possui uma função virtual void imprimirDados(), que tem seu funcionamento modificado na segunda classe.

```
void Pessoa::imprimirDados() {

std::cout << "Nome:" <<_nome << "\t CPF:"<< _cpf <<" \tEndereco:" << _rua << " \
tTelefone:" << _telefone << " \tData de Inicio:" << _dataInicio << std::endl;

}

void Paciente::imprimirDados() {

std::cout << "Nome:" <<_nome << "\tEndereco:"  << _rua << " \tTelefone:" << _telefone <<
"\tData de Inicio:"<< _dataInicio << "\tPsicologo Responsavel:" << _psicologoResponsavel
<< std::endl << "Queixa:" << _queixa << std::endl;

}
```

6 CONCLUSÃO

O desenvolvimento de softwares unidos ao sistema de gestão empresarial contribuem positivamente para melhoria do fluxo de informação entre as diversas células dentro de um sistema empresarial, otimizando o tempo produtivo e auxiliando tomadas de decisão. Os conceitos apresentados em sistema de informação se tornam fundamentais para aderência das necessidades da demanda em um programa computacional visto que

o mesmo, pode apresentar restrições de herança, polimorfismo entre as classes, os métodos adequados de acesso às listas do banco de dados entre outras funções. Quando bem implementadas, o programa apresenta legibilidade e rápido desempenho, e para isso, é necessário que a equipe de desenvolvimento esteja com as ideias bem alinhadas para que o trabalho proposto siga a mesma linha de raciocínio, e é daí que retiramos também a importância da modularização do programa que auxilia no processo construtivo do software.

Dado o exposto, observou-se que a possível aplicação do software na clínica Acolhida traria grandes benefícios para o processo gerencial entre funcionários e pacientes, visto as suas inter-relações e restrições conforme apontados no desenvolvimento da pesquisa. É possível observar que este é também o caminho adotado nas grandes corporações que buscam otimizar o lead time de transmissão de informação e processamento de dados entre os seus diversos setores, tornando os conhecimentos aplicados, a base para o desenvolvimento de softwares no mundo moderno.