UNIP EaD

Projeto Integrado Multidisciplinar

Cursos Superiores de Tecnologia

**MONITORAMENTO DE DIAGNÓSTICOS DA COVID-19**

Uma Aplicação em Linguagem C

TAUBATÉ – SP

2022

UNIP EaD

Projeto Integrado Multidisciplinar

Cursos Superiores de Tecnologia

**MONITORAMENTO DE DIAGNÓSTICOS DA COVID-19**

Uma Aplicação em Linguagem C

Orientador (a):

Profa. Dra. Vanessa Lessa

César Augusto Alves Rodolfo Eskildesen – RA 2224506

Rinaldo da Silva – RA 2223241

Curso: Análise e Desenvolvimento de Sistemas

Semestre: 2º

TAUBATÉ – SP

2022

Sumário

[1. INTRODUÇÃO 6](#_Toc120131297)

[2. ANÁLISE DE REQUISITOS 8](#_Toc120131298)

[2.1 Requisitos Funcionais 8](#_Toc120131299)

[2.2 Requisitos não funcionais 9](#_Toc120131300)

[3. MODELO DE CICLO DE VIDA 10](#_Toc120131301)

[3.1 Ciclo de Vida RAD 10](#_Toc120131302)

[3.2 Implementação da Metodologia no Desenvolvimento do Software 10](#_Toc120131303)

[3.3 Fluxogramas 11](#_Toc120131304)

[4. CODIFICANDO O APLICATIVO 14](#_Toc120131305)

[4.1 Definindo entidades 14](#_Toc120131306)

[4.2 Autenticação 14](#_Toc120131307)

[4.3 Rotinas utilitárias 14](#_Toc120131308)

[4.4 Validação do E-mail 15](#_Toc120131309)

[4.5 Leitura e Gravação dos Diagnósticos 16](#_Toc120131310)

[4.6 Extraindo um conceito oculto 16](#_Toc120131311)

[4.7 Princípios KISS e Clean Code 17](#_Toc120131312)

[4.8 Nomes significativos 18](#_Toc120131313)

[4.9 Funções curtas 18](#_Toc120131314)

[4.10 Compilando e rodando 18](#_Toc120131315)

[4.11 Manual do Desenvolvedor 19](#_Toc120131316)

[5. MANUAL DE INSTALAÇÃO E USO 20](#_Toc120131317)

[5.1 Instalação 20](#_Toc120131318)

[5.2 Login 20](#_Toc120131319)

[5.3 Cadastro do Paciente 21](#_Toc120131320)

[5.4 Consulta aos Dados dos Paciente do Grupo de Risco 24](#_Toc120131321)

[5.5 Validação de Dados 25](#_Toc120131322)

[6. QUALIDADE E TESTES DO SOFTWARE 26](#_Toc120131323)

[6.1 Testes Unitários 26](#_Toc120131324)

[6.1.1 Framework mínimo 26](#_Toc120131325)

[6.1.2 Compilando e rodando os testes 27](#_Toc120131326)

[6.2 Testes de Integração (Check-list) 27](#_Toc120131327)

[7. CONCLUSÃO 28](#_Toc120131328)

[REFERÊNCIAS 29](#_Toc120131329)

[ANEXO A – Check-list 30](#_Toc120131330)

RESUMO

O Projeto Integrado Multidisciplinar IV do curso de Análise e Desenvolvimento de Sistemas, aborda todas as etapas de desenvolvimento do software de monitoramento de diagnósticos da COVID-19, trazendo a análise de requisitos, o modelo de ciclo de vida escolhido, os fluxogramas, a codificação, os manuais do desenvolvedor e do usuário, e, por fim, os elementos que atribuem qualidade ao sistema.

**Palavras-chave:** Software, COVID-19, Requisitos, Fluxogramas, Codificação, Manuais, Qualidade.

ABSTRACT

The Integrated Multidisciplinary Project IV of the Systems Analysis and Development course addresses all stages of development of the COVID-19 diagnostic monitoring software, bringing the requirements analysis, the chosen lifecycle model, the flowcharts, the coding, the developer and user manuals, and, finally, the elements that attribute quality to the system.

**Keywords:** Software, COVID-19, Requirements, Flowcharts, Coding, Manuals, Quality.

INTRODUÇÃO

O objetivo deste Projeto Integrado Disciplinar (PIM) IV é integrar as disciplinas Engenharia de Software I e Linguagem e Técnicas de Programação (Linguagem C), com a finalidade de se produzir uma aplicação para Controle e Monitoramento de Diagnósticos da COVID-19.

A pandemia da COVID-19 trouxe uma grande demanda por novos recursos na área da saúde. Mas a demanda que será abordada neste projeto é referente a novas tecnologias, mais especificamente, software. Com um número exponencial de casos de infecção, foi observado que os métodos anteriores para o rastreamento e diagnóstico da doença não satisfaziam as atuais necessidades, dessa forma, faz-se necessário a criação de novos meios facilitadores, para que este trabalho seja feito de forma automática e rápida

Foi solicitado o desenvolvimento de um sistema para que seja possível o cadastro de pacientes com diagnóstico positivo para COVID-19, solicitando e gravando dados pessoais e de endereço. Desta forma, com base nos dados obtidos, mais especificamente a idade (calculada por meio da data de nascimento informada) e se existem ou não comorbidades, o sistema deverá identificar se o paciente faz parte do grupo de risco ou não. Se sim, os dados deste indivíduo serão gravados em um arquivo e listados juntamente com outros dados de outros diagnósticos de integrantes do grupo de risco.

Os desenvolvedores do projeto e do software realizaram uma análise dos requisitos do projeto e optaram pelo modelo RAD como ciclo de vida da aplicação. Percebeu-se que a criação de componentes, em especial, para delimitar as três entidades principais da aplicação, facilitaria bastante o desenvolvimento e a validação do projeto.

Foram construídos alguns fluxogramas para demonstrar o funcionamento dos componentes mencionados acima, bem como para explicitar o resultado final esperado, ou seja, a criação dos arquivos que conterão os diagnósticos e o pacientes diagnosticados como sendo do grupo de risco, ou seja, pacientes acima de 65 anos ou com comorbidades.

O capítulo 4 abrange como foi feita a codificação deste projeto, usando princípios ágeis e orientadores, como por exemplo, o KISS e o Clean Code. O objetivo foi criar um código simples, legível, de fácil entendimento e de fácil manutenção. Os arquivos e sub-rotinas são pequenos, desacoplados e coesos. Deu-se bastante atenção à validade das informações que serão digitadas, como por exemplo, datas, CPF, e-mail etc., através da construção de rotinas de validação meticulosamente planejadas. O desenvolvedor também poderá contar com um manual que explica em detalhes as entidades e sub-rotinas da aplicação.

No capítulo 5, encontra-se o manual de instalação e uso da aplicação. Através de uma interface bastante simples e fluida, o usuário fará a digitação dos dados solicitados e o sistema irá gravar os arquivos necessários, conforme os requisitos descritos no manual do PIM. Os processos da aplicação estão exemplificados em figuras que permitem ao usuário entender corretamente como atuar na operação do software.

Por fim, o capítulo 6 apresenta o que foi construído para garantir a qualidade e a integridade do software. Foi utilizado um framework mínimo para a construção de testes unitários automatizados, o que aumenta bastante a confiabilidade da aplicação. Providenciou-se também um check-list com testes de integração, para a validação de todas as funcionalidades programadas, bem como dos resultados gravados no arquivo texto.

ANÁLISE DE REQUISITOS

Este projeto define, através do manual, uma série de requisitos que foram estudados e analisados com bastante atenção pelos desenvolvedores com a finalidade de se produzir um software simples, leve e com a capacidade de produzir os resultados esperados.

A aplicação deveria ser construída utilizando a linguagem de programação C, utilizando a IDE gratuita Code::Blocks. São requisitos do sistema também, acesso protegido por meio de Login e Senha e que haja métodos de validação de dados, chegando o mais próximo possível de um sistema à prova de falhas, onde não ocorra a entrada de dados incorretos.

O sistema deve ser leve, de fácil instalação, acesso e uso, visando que será operado por profissionais da área da saúde, que não possuem nível alto de *expertise* em operação de sistemas, o que demanda também um treinamento prévio da equipe antes da implementação do sistema; o software também deve operar nos sistemas operacionais mais utilizados.

A qualidade e a integridade da aplicação também foram entendidas como requisitos implícitos. Dessa forma, dentro dos recursos oferecidos pela IDE, buscou-se formas de se codificar e executar testes unitários automatizados.

De acordo com a urgência que a pandemia e o crescimento dos casos estão trazendo, o sistema deverá ser desenvolvido e rapidamente implementado, com o prazo de poucas semanas para a entrega final do projeto.

1. Requisitos Funcionais

Os requisitos funcionais do sistema consistem em:

* Acesso protegido por Login e Senha;
* Cadastro de pacientes com entrada dos seguintes dados: Nome, CPF, Telefone, Endereço (rua, número, bairro, cidade, estado e CEP), Email, Data de Nascimento, Data do Diagnóstico e Comorbidades do Paciente;
* Validação dos dados inseridos;
* Cálculo da idade do paciente;
* Emissão de arquivo de texto listando CEP e a idade dos pacientes enquadrados no grupo de risco.

1. Requisitos não funcionais

Os requisitos não funcionais do sistema consistem em:

* Testes unitários e de integração para minimizar ao máximo as falhas;
* Requisitar o mínimo espaço de memória possível para sua instalação e uso;
* Sistema intuitivo, fácil de operar;
* Treinamento prévio dos usuários antes da implementação do software;
* Compatível com os sistemas operacionais mais utilizados;
* Prazo curto para desenvolvimento, existe urgência para a entrega.

MODELO DE CICLO DE VIDA

Ao analisar as expectativas e demandas, fica nítido que existe grande importância em uma entrega rápida da aplicação, porém não renunciando a um projeto funcional e de alta qualidade. Devido a estas razões, o ciclo de vida deste software claramente precisa ser um modelo ágil de desenvolvimento.

1. Ciclo de Vida RAD

O ciclo de vida escolhido foi o RAD (Rapid Application Development), modelo que atende muito bem os requisitos do projeto.

Conforme Cipullo (2021), o RAD é indicado quando o software tem como objetivo entregas rápidas, em torno de 60 a 90 dias, pois tem o foco voltado diretamente para a prototipagem do projeto, reduzindo grande tempo de planejamento. Portanto, encaixa-se perfeitamente em um dos requisitos principais dado o curto espaço de tempo disponível para o desenvolvimento.

Também com base nas características desta metodologia, como os objetivos do projeto estão bem definidos e contando com uma análise de requisitos muito clara e objetiva, é possível trabalhar com um desenvolvimento incremental dividindo cada função do software em módulos separados onde a cada sprint é adicionada e incrementada uma nova funcionalidade, facilitando assim um grande agente facilitador em um desenvolvimento rápido que é a reutilização de componentes/códigos prontos.

1. Implementação da Metodologia no Desenvolvimento do Software

Primeiramente foi feita a análise de requisitos juntamente com a compreensão e refinamento das informações obtidas. Em seguida, percebeu-se a existência de entidades, as quais foram denominadas: *Paciente*, *Endereço* e *Diagnóstico*. Depois foi feita a modelagem do fluxo de implementação no projeto, através dos fluxogramas, onde foram estruturadas todas as funções do sistema sequencial de forma gráfica e foi possível ter melhor compreensão do processo e do tratamento de dados.

Por conseguinte, foram divididos entre os dois desenvolvedores do projeto os módulos que cada um seria responsável por desenvolver, a serem desenvolvidos de forma sequencial, para que a cada sprint pudessem ser integrados, validados e testados.

Por se tratar de um software razoavelmente simples, com funcionalidades muito parecidas com a que encontramos em diversos outros programas, foi possível fazer um bom reuso do código-fonte, a maior parte nas rotinas utilitárias para tratamento de dados, como para converter datas para o padrão dia/mês/ano, identificar anos bissextos, identificar existência de caracteres ou números, transformar para caracteres maiúsculos, bem como rotinas de validação para e-mail, CPF, CEP etc.

Somadas todas essas práticas, foi possível desenvolver um projeto de qualidade em tempo hábil para cumprir com todos os requisitos.

1. Fluxogramas

Após definidas as entidades, foram modelados, utilizando a ferramenta online Miro (miro.com), fluxogramas com as rotinas e sub-rotinas do sistema, com o objetivo de esclarecer tanto para o cliente quanto para o desenvolvedor as funcionalidades do software.

Seguem abaixo os fluxogramas de cada uma das entidades (Paciente, Endereço e Diagnóstico).

Figura 1 – Fluxograma da entidade Paciente

Diagrama

Descrição gerada automaticamente

Fonte: Autoria Própria.

Figura 2 – Fluxograma da entidade Endereço

Diagrama

Descrição gerada automaticamente

Fonte: Autoria Própria.

Figura 3 – Fluxograma da entidade Diagnóstico

Diagrama

Descrição gerada automaticamente

Fonte: Autoria P

rópria.

CODIFICANDO O APLICATIVO

1. Definindo entidades

Percebeu-se, ao se levantar os requisitos do projeto, a existência de três entidades ou objetos principais, os quais poderiam ser implementados, testados e validados separadamente. São eles: *Endereço*, *Paciente* e *Diagnóstico*. Usando o padrão de projetos Composição, cada entidade é utilizada na relação 1 para 1 pela entidade seguinte.

Para cada objeto, foi determinado que seriam criados um arquivo header .h e um arquivo .c, que conteriam as definições da entidade (struct) e as rotinas de leitura e validação da entidade, respectivamente. Além disso, no arquivo .c referente à entidade Diagnóstico, seriam colocadas as rotinas e regras de negócio para se efetuar a gravação dos arquivos binário e texto requeridos.

1. Autenticação

Para que o usuário pudesse se autenticar com usuário e senha, foi criado um módulo (arquivos autenticacao.h e autenticacao.c) coeso e sem acoplamento, contendo toda a especificação necessária. A sub-rotina exportada no arquivo header permite a passagem de um parâmetro char \*, o qual receberá o login do usuário que se autenticou no sistema.

Não existe um repositório ou banco de dados, para que se pudesse buscar o usuário e senha digitados. Dessa forma, optou-se por criar um vetor de strings, com 3 usuários e senha pré-estabelecidos, com os quais o aplicativo poderá ser utilizado.

1. Rotinas utilitárias

Foi necessário um intenso desenvolvimento de sub-rotinas para tratamento de strings, datas, bem como para validar os dados digitados pelos usuários. Essas rotinas foram colocadas na biblioteca utils (utils.h e utils.c).

Novamente, por ser desenvolvido de modo coeso e sem acoplamento, permitiu um maior reaproveitamento de código, bem como facilitou bastante também a escrita de testes automatizados.

Trata-se do maior arquivo do código-fonte, mesmo assim, contendo apenas cerca de 260 linhas. Todas as rotinas foram devidamente comentadas para gerar o manual do desenvolvedor, descrito mais adiante nesse mesmo capítulo. Foram dados os devidos créditos a sites da internet onde os programadores se basearam para escrever algumas rotinas.

As rotinas referentes ao cálculo da idade e validação das datas digitadas foram originalmente escritas para usar a biblioteca time.h, da linguagem C. Porém, como as datas anteriores a 1970 não eram corretamente calculadas pela rotina mktime(), foi necessário escrever duas sub-rotinas, em especial, para calcular a idade de uma pessoa. [[1]](#footnote-1)Manteve-se, entretanto, a utilização da struct tm, proveniente da biblioteca time.h.

1. Validação do E-mail

Desde o início, tomou-se uma grande preocupação de se validar o mais precisamente possível os dados digitados pelos usuários. Embora não seja um dado obrigatório no cadastro do paciente, não se desejava que os usuários digitassem endereços de e-mail, CPF, ou datas inválidas.

Por experiência prévia dos desenvolvedores, imaginava-se ser possível a validação através de Expressões Regulares. Porém, a biblioteca regex.h apenas se encontra disponível na linguagem C++. Assim sendo, tornou-se necessário “destrinchar” a string digitada pelo usuário.

Após uma pesquisa na internet, encontrou-se diversas regras de validação, que foram implementadas na biblioteca email (email.h e email.c). Como foram escritas diversas sub-rotinas de validação das partes (local e domínio) de um e-mail, optou-se por agrupar tudo numa única biblioteca, expondo apenas uma única função que poderá ser chamada de qualquer outra parte da aplicação. As demais rotinas são estáticas no arquivo email.c.

1. Leitura e Gravação dos Diagnósticos

A biblioteca diagnostico (diagnostico.h e diagnostico.c) faz a implementação da entidade Diagnóstico, além de executar as rotinas de gravação dos dois arquivos requeridos para o aplicativo.

A rotina lerDiagnosticos(char \*usuario) implementa um loop de leitura para o Diagnóstico, o qual por sua vez abre outros loops para digitação dos dados dos objetos subsequentes.

O usuário finaliza a aplicação ao digitar “sair”, por ocasião da leitura da data do diagnóstico. Se o usuário não digitar nada para essa data, será assumida a data do sistema.

1. Extraindo um conceito oculto

Quando o paciente é do grupo de risco, deverá ser gravado um arquivo-texto que conterá o CEP e a idade deste paciente. Muitos desenvolvedores apenas colocarão um *if* dentro do código para realizar a checagem e a gravação, como segue abaixo:

|  |  |  |
| --- | --- | --- |
|  | static void verificarGrupoRisco(Paciente \*paciente)  {      int idade = calcularIdade(paciente->data\_nascimento);      if (idade >= IDADE\_RISCO || strlen(paciente->comorbidades)) {          printf("Gravando informações de paciente do grupo de risco...\n");          ... |  |

De acordo com Evans (2009, p. 17), uma importante regra de negócio (linha 5) está oculta no “emaranhado” do código. Isso poderia ocasionar pelo menos duas situações indesejáveis:

1. Um especialista de negócio pode não conseguir ler e verificar esta regra, mesmo com a ajuda de um desenvolvedor.
2. Seria difícil para uma pessoa técnica, mas não da área de negócios, conseguir ligar o texto de requisitos e o código fonte.

Para resolver esse problema, a solução foi extrair a regra de negócio numa sub-rotina responsável por fazer a verificação necessária. O código fica mais simples e elegante, de maneira que, um especialista, um desenvolvedor ou um técnico conseguem facilmente identificar a regra de negócio. Segue abaixo uma pequena extração do código fonte:

|  |  |  |
| --- | --- | --- |
|  | static int ehGrupoRisco(Paciente \*paciente, int idade)  {      return idade >= IDADE\_RISCO || strlen(paciente->comorbidades);  }  static void verificarGrupoRisco(Paciente \*paciente)  {      int idade = calcularIdade(paciente->data\_nascimento);      if (ehGrupoRisco(paciente, idade)) {          printf("Gravando informações de paciente do grupo de risco...\n");          ... |  |

1. Princípios KISS e Clean Code

“Keep it simple, stupid (KISS) é um princípio de design que afirma que designs e/ou sistemas devem ser o mais simples possível” (Keet It Simple, Stupid (KISS), [s.d.], a tradução é nossa). Não se trata de um ofensa a ninguém, mas, antes, o princípio KISS enfatiza a necessidade de se manter o design de qualquer tipo de projeto “estupidamente” simples.

Além do KISS servir de guia para os desenvolvedores, foram adotadas também as boas práticas do Clean Code. O autor Robert C. Martin, no livro Clean Code, cita Grady Booch, autor do livro Object Oriented Analysis and Design with Applications, o qual define Clean Code ou Código Limpo da seguinte forma:

“O código limpo é simples e direto. Código limpo parece uma prosa bem escrita. Código limpo nunca obscurece a intenção do designer, mas é cheio de abstrações nítidas e linhas diretas de controle.” (MARTIN, 2009, p. 7, tradução nossa).

De modo geral, os desenvolvedores procuraram desenvolver um código eficiente, mas sem ser insuficiente na legibilidade e ao mesmo tempo, sem deixar de usar funcionalidades da linguagem C, como por exemplo, a atribuição e o uso de variáveis *inline,* declaração de variáveis no bloco ou no momento da utilização e o operador ternário (?), etc.

Além disso, no arquivo utils.h, foram criadas algumas macros para facilitar o entendimento e a digitação repetitiva de certas sequências de código.

Nas próximas subseções, é exposto como se procurou aplicar muitas das boas práticas de programação de Clean Code, ao escrever o aplicativo.

1. Nomes significativos

Com exceção de variáveis contadoras ou de raras vezes em que fica muito simples, prático e intuitivo, declarar, dentro de um pequeno bloco de código, algo como *Paciente p* ou *trim(char \*s),* os nomes de variáveis e de sub-rotinas são expressos de maneira significativa e pronunciável, como por exemplo: idade, nome, paciente, calcularIdade(), dataSistema(), etc.

Evitou-se também o uso da língua inglesa, exceto em algumas funções, nas quais o uso em inglês fica mais desejável, como: trim(), strofchar(), openFile(), etc., bem como nas macros da utils.h.

Vale a apena destacar que não existe nem uma única variável global em toda a aplicação.

1. Funções curtas

Existem duas regras fundamentais para a escrita de funções. São as seguintes: “A primeira regra das funções é que elas devem ser pequenas. A segunda regra das funções é que devem ser menores que isso”. (MARTIN, 2009, p. 34, tradução nossa).

Procurou-se seguir à risca essa máxima no desenvolvimento do aplicativo. Todas as funções, ou sub-rotinas, são pequenas e realizam apenas uma única tarefa.

1. Compilando e rodando

Com a criação da aplicação para execução de testes unitários (testes.c), acabou ficando inviável compilar e rodar pelo Code::Blocks. Mesmo assim, em alguns casos, como para depuração, usou-se o simples artifício de renomear a função *int main()* em testes.c para qualquer outro nome.

Para efetivamente compilar e rodar, tanto o aplicativo, como os testes unitários, criou-se um arquivo *make* (makefile) e dois arquivos .bat que o executam. O arquivo testes\_unitarios.bat além de compilar, também executa os códigos unitários. Para que o *make* funcione corretamente, o gcc e o mingw32-make devem estar no PATH do Windows.

Na janela de prompt, na mesma pasta onde estiver o código fonte, ou ainda, pelo Windows Explorer, deve-se compilar e rodar, utilizando o arquivo compilar.bat, como mostrado abaixo na figura abaixo:

Figura 4 – Compilação

Texto

Descrição gerada automaticamente

Fonte: Autoria Própria.

1. Manual do Desenvolvedor

Como o código foi escrito utilizando as boas práticas do Clean Code, não se torna necessário comentar o código fonte, visto que o código é autoexplicativo.

Porém, a fim de usar a ferramenta DoxyGen (https://doxygen.nl/), de geração automatizada de documentação do código-fonte, foram inseridos comentários, com uma formatação própria e específica, nos arquivos .h do código-fonte.

A manutenção e revisão deste manual é muito importante para que o desenvolvedor possa ter uma explicação breve e objetiva sobre a utilização das diversas partes da aplicação, bem como para otimizar a reutilização de código, evitando que o desenvolvedor “reinvente a roda”, escrevendo código em duplicidade.

MANUAL DE INSTALAÇÃO E USO

1. Instalação

O software DiagCovid não requer instalação, basta que o usuário acesse a pasta onde está localizado o arquivo executável DiagCovid.exe. Esta pasta pode estar compactada em formato RAR ou ZIP. Neste caso, para executar o programa, o usuário deverá ter previamente instalado um programa extrator de arquivos, para que assim faça a descompactação e possa rodar a aplicação.

1. Login

Na tela de Login é solicitado ao usuário apenas duas informações Login e Senha. A autenticação deste usuário depende da correspondência dos dados inseridos com os dados contidos no código fonte do arquivo autenticacao.c. Na função *autenticar(),* existem 3 usuários e senhas pré-definidos.

Seguem abaixo capturas da tela de login para demonstrar o funcionamento:

Figura 5 – Login autenticado

Texto

Descrição gerada automaticamente

Fonte: Autoria Própria.

Figura 6 – Login não autenticado

Texto

Descrição gerada automaticamente

Fonte: Autoria Própria.

Quando o usuário é autenticado, o sistema segue automaticamente com o cadastro do paciente. Se o usuário errar 3 vezes consecutivas, será desconectado da aplicação.

1. Cadastro do Paciente

Nesta tela é solicitado primeiramente que usuário informe a data do diagnóstico, onde pode optar por digitar a data, ou caso o cadastro esteja sendo feito no mesmo dia do diagnóstico, pode-se apenas pressionar a tecla Enter que o sistema preencherá automaticamente com a data do dia atual.

Figura 7 – Data do diagnóstico

Texto

Descrição gerada automaticamente

Fonte: Autoria Própria.

Após informada a data, o sistema segue para o cadastro de dados pessoais do paciente: nome, CPF, Data de Nascimento, Telefone, E-mail, Comorbidades.

Deve-se ter bastante atenção ao preencher o campo de comorbidades, qualquer carácter inserido neste será entendido como uma sinalização positiva, caso o paciente não tiver nenhuma comorbidade deve-se deixar este campo em branco.

Sendo assim, pacientes com idade acima de 65 anos e/ou acometidos por comorbidades estarão automaticamente inseridos no Grupo de Risco do sistema e terão seu CEP e Idade coletados por ele.

Após a inserção destes dados acontecerá a validação deles, caso válidos será solicitada a entrada dos dados de endereço do paciente e caso não estejam de acordo, será solicitada uma nova entrada. Seguem capturas de tela demonstrando este funcionamento:

Figura 8 – Dados pessoais inválidos

Texto

Descrição gerada automaticamente

Fonte: Autoria Própria.

Figura 9 – Dados pessoais válidos

Texto

Descrição gerada automaticamente

Fonte: Autoria Própria.

Após a entrada dos dados pessoais, o sistema aguardará a inserção dos dados de endereço, onde, após inseridos, passarão por validação, e assim como os dados pessoais, caso inválidos, o sistema solicitará nova entrada, e caso válidos, o sistema gravará o diagnóstico. Seguem abaixo capturas de tela demonstrando estes cenários:

Figura 10 – Dados de endereço inválidos

Texto

Descrição gerada automaticamente

Fonte: Autoria Própria.

Figura 11 – Dados de endereço válidos

Texto

Descrição gerada automaticamente

Fonte: Autoria Própria.

Ficando a critério do usuário finalizar a operação ou continuar cadastrando novos pacientes.

1. Consulta aos Dados dos Paciente do Grupo de Risco

Caso o paciente cadastrado tenha mais de 65 anos e/ou tenha comorbidades o sistema entenderá que ele faz parte do grupo de risco, e por conseguinte, gravará em um arquivo CSV seu respectivo CEP e Idade.

Figura 12 – Cadastro de paciente do grupo de risco

Texto

Descrição gerada automaticamente

Fonte: Autoria Própria.

Quando cadastrado um paciente do grupo de risco, o sistema emitirá a mensagem “Gravando informações de paciente do grupo de risco...” e enviará o CEP e a idade do mesmo para o arquivo casos-grupo-risco.csv, o qual está localizado dentro do diretório principal da pasta DiagCovid, e para consultar os dados destes pacientes deve-se abrir este arquivo, o que pode ser feito utilizando o Microsoft Excel ou qualquer editor de texto. Segue abaixo uma demonstração da consulta destes dados utilizando o Excel:

Figura 13 – Consulta aos dados de paciente do grupo de risco.

Interface gráfica do usuário, Aplicativo, Tabela, Excel

Descrição gerada automaticamente

Fonte: Autoria Própria.

1. Validação de Dados

Para a validação dos dados digitados, não é feita uma validação após cada input do usuário.

Ao invés disso, é feita uma validação da entidade como um todo. Se houver erros na validação, tais são exibidos na tela, permitindo ao usuário reiniciar a digitação, com a finalidade de corrigir os dados incorretos. Tal ciclo se repete até que o usuário tenha digitado todos os dados corretamente, ou ao indicar que deseja cancelar a digitação.

Ao terminar a digitação do Diagnóstico e gravação nos arquivos necessários, reinicia-se novamente o ciclo, somente terminando quando o usuário digitar "sair".

QUALIDADE E TESTES DO SOFTWARE

Testar um software é um processo pelo que se determina se um produto atingiu suas especificações e funcionou corretamente para a finalidade para o qual projetado. Se falhas forem encontradas, tais podem ser corrigidas pela equipe de desenvolvimento, antes da entrega ao cliente. Pode-se dizer então tal processo se trata de uma natureza “destrutiva”, pois aumenta a confiança no software através da exposição de seus problemas, obviamente, antes da entrega final (NETO, 2007).

Foram feitos dois tipos de testes no software: unitários e de integração. Os testes unitários foram realizados de forma automatizada. Os testes de integração foram feitos de modo manual pelos próprios desenvolvedores, através da utilização de um checklist das características esperadas do funcionamento do software.

1. Testes Unitários
2. Framework mínimo

Depois de uma extensa pesquisa na internet, em busca de uma biblioteca ou plugin para ser acoplado ao CodeBlocks, a qual se mostrou insatisfatória, localizou-se no site https://jera.com/techinfo um framework mínimo, mas que se demonstrou bastante adequado ao projeto. Trata-se do framework MinUnit.[[2]](#footnote-2)

Trata-se simplesmente de um arquivo .h, o qual implementa duas macros C para realizar as asserções e a execução da sub-rotina de testes. Os desenvolvedores do projeto adicionaram uma terceira macro para especificar o tipo de retorno da sub-rotina (static char\*). Quando uma asserção falha, é exibida na tela uma mensagem de erro customizada para o teste específico que falhou. Se todos os testes são executados com sucesso, aparece para o desenvolvedor na tela, a mensagem “ALL TESTS PASSED”.

Assim sendo, foi incluído no projeto o arquivo minunit.h e o arquivo testes.c com a finalidade de executar testes automatizados das rotinas consideradas essenciais para o funcionamento do software.

Não foi a usada a metodologia TDD (Test Driven Development) para a construção das funcionalidades, nem dos testes unitários. As rotinas de testes automatizados foram escritas gradualmente conforme o desenvolvimento da aplicação.

1. Compilando e rodando os testes

Para facilitar para o desenvolvedor, na mesma pasta do código fonte, existe um arquivo testes\_unitarios.bat, o qual realiza a recompilação do testes.exe e a sua subsequente execução.

Segue abaixo, na figura um exemplo da execução dos testes unitários:

Figura 14 – Teste unitário

Texto

Descrição gerada automaticamente

Fonte: Autoria Própria.

1. Testes de Integração (Check-list)

O formulário do check-list está exposto no Anexo A deste trabalho. O check-list deve ser assinado pelo analista de testes e pelo gerente do desenvolvimento. O aplicativo deve ser reiniciado tantas vezes quanto necessário, para a realização de todos os passos necessários.

CONCLUSÃO

Com base no estudo das matérias disponibilizadas neste bimestre e por meio de pesquisas, foi possível a elaboração de um software utilizando a linguagem C, onde o desenvolvimento deste foi um grande ensejo para a exploração dos conteúdos de Engenharia de Software e Linguagens e Técnicas de Programação para além daquilo visto e disponibilizado em aula.

Além disto, foi observado também que apesar da Linguagem C ser considerada de alto nível, e ter atendido os requisitos do sistema, ela não parece ser a ideal para o desenvolvimento de projetos como este atualmente, criando algumas barreiras para o software e dificuldades para o desenvolvedor que acabam por impedir o sistema de atingir um padrão mais alto de qualidade, principalmente no que tange à estética e à experiência do usuário. Parece um tanto estranho, numa época em que a maioria das pessoas está acostumada a interfaces gráficas ricas e avançadas, voltar a trabalhar com sistemas num padrão “console” ou “DOS”.

A escolha do RAD, como modelo de ciclo de vida para este projeto, mostrou-se ser bastante assertiva e vantajosa. O desenvolvimento separado das entidades e dos componentes facilitou bastante para que os desenvolvedores alcançassem as expectativas referentes ao cumprimento dos requisitos do PIM.

Os desenvolvedores cogitaram adotar o modelo ágil Extremme Programming (XP), mas percebeu-se que a ausência da figura do “cliente” e que, pelo fato da equipe ser composta por apenas dois desenvolvedores, terminaria por descaracterizar o modelo em si.

Apesar de ser um projeto relativamente pequeno, os desenvolvedores procuraram aplicar boas práticas na escrita do código-fonte. Talvez, outros desenvolvedores optassem por uma abordagem mais direta e mais rápida, como o próprio modelo ágil XP orienta, no sentido de resolver o problema “de agora”. A ênfase dada na validação dos dados talvez não fosse tão necessária, mas permitiu aos desenvolvedores explorarem e conhecerem melhor a linguagem de programação C.

REFERÊNCIAS

EVANS, E. **Domain-Driven DESIGN: Atacando as Complexidades no Coração do Software.** 2a edição revisada. [s.l.]: Alta Books, 2009.

**Keet It Simple, Stupid (KISS).** Interaction Design Funcation. [s.d.], Disponível em <<https://www.interaction-design.org/literature/topics/keep-it-simple-stupid#:~:text=Keep%20it%20simple%2C%20stupid%20(KISS)%20is%20a%20design%20principle,of%20user%20acceptance%20and%20interaction>.> Acessado em 9 nov. 2022.

MARTIN R. C. **Clean Code: A Handbook of Agile Software Craftsmanship*.*** [s.l.]: Person Education Inc, 2009.

CIPULLO, G. **RAD: VOCÊ SABE COMO FUNCIONA O DESENVOLVIMENTO ÁGIL DE APLICAÇÕES? Viasoft Korp**, 8 jul. 2021. Disponível em: <https://www.korp.com.br/rad-voce-sabe-como-funciona-o-desenvolvimento-agil-de-aplicacoes/#:~:text=funcionalidades%20dos%20programas.-,Afinal%2C%20o%20que%20%C3%A9%20RAD%3F,mais%20curtas%20e%20prototipagem%20%C3%A1gil.>. Acesso em: 8 nov. 2022.

‌BUENO, João**. Análise e projetos de sistemas.** [ca. 2010]. Apresentação do Power Point. Disponível em: <http://docente.ifsc.edu.br/joao.augusto/MaterialDidatico/2018-1/Análise%20e%20Projeto%20de%20Sistemas/Levantamento%20dos%20Requisitos.pdf>. Acesso em: 11 nov. 2022.

ANEXO A – Check-list

|  |  |  |
| --- | --- | --- |
| Funcionalidade | Atividade de Teste | Verificado |
| Digitação do diagnóstico | Digitar sair. Verificar se a aplicação será encerrada. |  |
|  | Não digitar data. Verificar se o aplicativo assumirá a data do sistema como padrão e passará a pedir os dados do paciente. |  |
|  | Digitar a data 25/02/2020. A data será considerada como inválida por ser uma data anterior ao 1o caso de COVID no Brasil |  |
|  | Digitar uma data posterior à  data de 25/02/2020 e menor ou igual à data do sistema. Verificar se o aplicativo pedirá os dados do paciente. |  |
| Digitação do paciente | Não digitar nenhum campo: verificar se o programa pede para digitar novamente os dados. |  |
|  | Digitar o CPF com caracteres não-numéricos ou com um tamanho diferente de 11 dígitos e verificar se a mensagem de erro é exibida. |  |
|  | Digitar as seguintes datas de nascimento que serão consideradas inválidas pelo software: 32/01/2022, 29/02/2022, 01/13/2022, 01/01/2100. Verificar se a mensagem de erro é exibida. |  |
|  | Digitar os seguintes e-mails inválidos: aa, a@a@a.com, -a@teste.com,  a@teste, 12345@teste.com.  Verificar se a mensagem de erro é exibida corretamente. |  |
|  | Deixar o nome sem preenchimento e digitar um e-mail válido, por exemplo: email@provedor.com.  Nenhuma mensagem de erro deve ser exibida. |  |
|  | Preencher todos os campos corretamente, utilizando a data de nascimento 01/01/1945. Verificar se o aplicativo pedirá os dados do endereço. |  |
| Digitação do endereço | Não digitar nenhum campo: verificar se o programa pede para digitar novamente o endereço |  |
|  | Digitar bairro e cidade com 1 caracter apenas e verificar se o programa exibe as mensagens de erro correspondentes. |  |
|  | Digitar um estado do Brasil inválido, por exemplo SS e verificar se a mensagem de erro é exibida. |  |
|  | Digitar um CEP com caracteres não-numéricos ou com um tamanho diferente de 8 dígitos: verificar se a mensagem de erro é exibida. |  |
| Finalização | Informar todos os campos corretamente: verificar se o aplicativo gravará o diagnóstico (arquivo diagnosticos.dat) e se incluirá no arquivo casos-grupo-risco.csv o CEP e a idade do paciente. |  |

Analista de testes:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Data:   /  /

Gerente de Desenvolvimento:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Data:   /  /

1. https://developercommunity.visualstudio.com/t/mktime-does-not-support-dates-before-jan-1-970/1208504. Acesso em: 8 nov. 2022. [↑](#footnote-ref-1)
2. https://jera.com/techinfo/jtns/jtn002. Acesso em: 3 nov. 2022. [↑](#footnote-ref-2)