UNIVERSIDADE ESTADUAL DO RIO GRANDE DO SUL

ENGENHARIA DE COMPUTAÇÃO

FERNANDA SILVA FERREIRA

ENGENHARIA REVERSA PARA GERAÇÃO DE DOCUMENTAÇÃO DE SOFTWARES LEGADOS

GUAÍBA

2021

FERNANDA SILVA FERREIRA

ENGENHARIA REVERSA PARA GERAÇÃO DE DOCUMENTAÇÃO DE SOFTWARES LEGADOS

Trabalho de Conclusão de Curso. Apresentado como requisito parcial para obtenção do título de Engenheira de Computação na Universidade Estadual do Rio Grande do Sul.

Orientadora: Prof. Dra. Margrit Reni Krug

GUAÍBA

2021

FERNANDA SILVA FERREIRA

ENGENHARIA REVERSA PARA GERAÇÃO DE DOCUMENTAÇÃO DE SOFTWARES LEGADOS

Trabalho de Conclusão de Curso. Apresentado como requisito parcial para obtenção do título de Engenheira de Computação na Universidade Estadual do Rio Grande do Sul.

Aprovada em ..../..../.....

BANCA EXAMINADORA:

Prof. Dr. Celso Maciel da Costa
Universidade Estadual do Rio Grande do Sul

Prof. Dra. Fabrícia Damando Santos
Universidade Estadual do Rio Grande do Sul

Prof. Dra. Margrit Reni Krug
Universidade Estadual do Rio Grande do Sul
Orientadora

AGRADECIMENTOS

Agradeço, em primeiro lugar, à minha família, pelo suporte, força e exemplo de vida proporcionado a mim. Agradeço também aos meus amigos e amigas, que me acompanharam nesta jornada dedicando apoio e caminharam lado a lado, formando laços que jamais serão quebrados. Durante este período como estudante na UERGS, pude participar ativamente nas atividades oferecidas de forma extracurricular. Tais atividades me moldaram como estudante, cidadã e profissional que hoje sou. Por isto, agradeço também aos professores e corpo técnico da unidade Guaíba da UERGS, pelo aprendizado proporcionado e atenção dada. Tenho orgulho em dizer que faço parte desta universidade.

RESUMO

A existência de softwares legados faz parte da realidade atualmente, entretanto, sua documentação descritiva pode ficar defasada ao longo do tempo ou pode não inexistir. Diante deste problema, é possível utilizar dos conceitos de engenharia reversa para criar um sistema que, realize a análise do código-fonte do software legado e gere uma documentação atualizada dele. No desenvolvimento deste sistema, utilizou-se a metodologia de pesquisa de natureza prática, com o intuito de validar a ideia aqui sugerida. Tal sistema é capaz de identificar, no código-fonte analisado, pontos relevantes para gerar uma documentação alto nível de abstração do software legado analisado. Como resultado, foi possível demonstrar a possibilidade de solução do problema através da aplicação da definição de engenharia reversa.

Palavras-chave: Engenharia Reversa. Softwares Legados. Documentação. Código-Fonte.

ABSTRACT

The existence of legacy software is part of reality today, however, its descriptive documentation may become outdated over time or may not exist. Given this problem, it is possible to use reverse engineering concepts to create a system to performs the analysis of the source code of the legacy software and generates an updated documentation. In the development of this system, a practical research methodology was used, in order to validate the idea suggested here. Such system is able to identify, in the source code to be analyzed, relevant points to generate a high level abstraction documentation of the analyzed legacy software. As a result, it was possible to demonstrate the possibility of solving the problem by applying the reverse engineering definition to it.

Keywords: Reverse Engineering. Legacy Software. Documentation. Source Code.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Manutenções em softwares legados são amplamente utilizadas, devido a rápida evolução das tecnologias existentes na área de software. Tais manutenções demandam conhecimento completo do funcionamento e dos requisitos do sistema, para então poder-se iniciar as correções necessárias. Este conhecimento pode vir da experiência prévia daqueles que trabalham desenvolvendo o sistema ou de documentos detalhados do projeto. A inexistência deste tipo de conhecimento pode dificultar, ou até mesmo, elevar os custos de manutenção de um software, como aponta Peres et al. (2003).

As documentações podem se perder ou não serem atualizadas com frequência, e assim tornarem-se defasadas, o que dificulta os ajustes e as consultas sobre o sistema. Segundo Sommerville (2011), os sistemas devem estar em constantes mudanças durante sua vida útil, seja para adaptar-se à uma nova plataforma, implementar uma nova funcionalidade ou atender a uma nova necessidade dos usuários.

Uma forma de resolver este problema é através da utilização da engenharia reversa, definido como um processo de recuperação do projeto a partir de seu código-fonte. Ferramentas são utilizadas para extrair informações sobre a arquitetura e procedimentos do projeto, e então gerar uma documentação abstrata do projeto. (COSTA, 1997).

1.1 TEMA E PROBLEMA

A engenharia de software permite a organização e planejamento de um software, iniciando na compreensão do problema, passando pela sua concepção até chegar em

sua utilização e futuras atualizações. Um bom planejamento deve utilizar ferramentas para registrar decisões e requisitos definidos no início do projeto e ao longo de sua concepção, estes registros poderão ser utilizados durante todo o andamento do projeto. O conceito de engenharia de requisitos, ação importante da engenharia de software, auxilia na concepção e organização de tais registros do projeto. (PRESSMAN, 2011).

Os softwares mais antigos, denominados de softwares legados, ainda estão em uso em grandes empresas, pois geralmente atendem às necessidades de seus usuários e modelos de negócio. Conforme dito por Pressman (2011), softwares legados possuem baixa qualidade em termos de engenharia de software moderna. Tal qualidade pode se dar devido a projetos não expansíveis, código intrincado, documentação pobre ou inexistente, casos de testes não arquivados, histórico de modificações mal administrado, entre outros pontos elencados por ele. Portanto, por conta de uma possível complexidade alta e um risco alto envolvendo uma atualização ou refatoração, estes softwares são mantidos da maneira em que se encontram, realizando-se apenas manutenções no código a fim de mantê-lo operacional.

Dentro de empresas que desenvolvem e mantém sistemas legados, é possível perceber a falta de conhecimento de aspectos importantes para a manutenção da vida útil do software. Costa (1997), afirmo que documentos descritivos e diagramáticos sobre o funcionamento, as regras, as restrições e os demais requisitos de um sistema são de suma importância para a criação e manutenção de um sistema.

Porém, ao longo do tempo e ao decorrer das mudanças realizadas, tais documentos ficam defasados, seja por falta de organização do gerente de projetos, por falta de tempo ao realizar uma entrega ou pela confiança por vezes empregada pelos desenvolvedores do projeto. Pode não parecer importante de início, mas este

detalhe de atualizar a documentação de um sistema pode impactar negativamente no futuro, atrasando o aprendizado de novos responsáveis pelo sistema ou atraso na entrega de novas funcionalidades e atualizações. (MOURA, 2008).

1.2 OBJETIVOS

Visando um aumento de desempenho e economia de tempo durante uma manutenção de software ou qualquer outra ação relacionada, uma documentação atualizada sobre seus requisitos e funcionalidades pode ajudar. Esta atualização das ferramentas de registros pode ser feita de forma manual, com uma atualização gradativa durante as modificações aplicadas, com uma ampla investigação do sistema ou com o agrupamento de diferentes fontes de informações sobre ele. Como desvantagem em relação a esta atualização manual está a perda de tempo e a má utilização do tempo dos membros capacitados do time. Assim sendo, uma ferramenta que possa gerar esta documentação atualizada do sistema, de forma automatizada, pode ser útil.

1.2.1 Objetivo Geral

A documentação inexistente ou fraca de um software legado pode ser solucionada utilizando o conceito de engenharia reversa, em que, a partir do código principal do sistema é extraído informações relevantes para gerar textos, fluxogramas e diagramas.

Com base nisto, este trabalho teve como objetivo geral a criação de um software para analisar códigos-fonte de sistemas legados para gerar automaticamente as documentações deste.

A atualização pode ser feita periodicamente ou conforme demanda determinada pelos responsáveis do projeto, visando sempre facilitar a visualização do sistema e o processo diário de desenvolvimento e manutenção de softwares legados.

Em suma, o desenvolvimento do software proposto tem como função principal acessar e ler o código-fonte de um sistema, organizar suas ações, para então traduzi-las em forma de textos abstratos. Possibilitando assim, uma rápida leitura dos requisitos do sistema, além de facilitar o seu acesso a todos os membros responsáveis pelo sistema.

1.2.2 Objetivos Específicos

Para alcançar o objetivo geral deste trabalho pode-se destacar os seguintes objetivos específicos:

- Identificar as falhas de registros de informações do sistema ocorridos ao longo de sua criação e manutenção. Em certos casos, onde não há nenhum tipo de registro conhecido sobre o sistema, tornou-se necessário trabalhar sem uma base, utilizando todo o código em linguagem de programação do sistema. Uma vez tendo algum tipo de informação registrado, possibilitou a aplicação do software desenvolvido apenas em partes necessárias do sistema analisado. A decisão do local da aplicação do software proposto pode ser feita por qualquer membro responsável pelo projeto;
- Analisar o código-fonte do software legado para extrair as principais informações e o fluxograma realizado durante seu funcionamento;
- Realizar a leitura do código-fonte do sistema para detectar as ações por ele realizadas, a fim de formar seu processo, o qual refere-se ao núcleo

do trabalho em questão. Uma boa detecção das funções realizadas garante a eficácia do software, possibilitando uma boa confiabilidade;

- Modelar a ferramenta possibilitando a execução de seu escopo de forma leve e rápida. A investigação da linguagem de programação utilizada durante a arquitetura do projeto garante seu melhor aproveitamento durante a fase de programação, tendo sempre em mente o acesso e inspeção do código-fonte do sistema a ser analisado. O desenvolvimento da ferramenta proposta, feita de forma experimental e aplicada em softwares de código aberto, disponíveis para uso livremente, para fins de prova de conceito;
- Gerar a documentação completa e confiável sobre o software, sendo utilizado textos de forma abstrata. Como a documentação é o artefato que é visto por todos os membros interessados na funcionalidade do sistema, torna-se importante que seja escrita de forma clara e concisa, além de ser disponibilizada em local de fácil acesso por todos, e;
- Validar a ferramenta proposta, através da comparação da documentação gerada com a já existente e com outras informações pertinentes já existentes do sistema analisado.

1.3 JUSTIFICATIVA

A documentação gerada, através deste trabalho, teve como objetivo auxiliar os profissionais envolvidos no projeto em andamento, sendo estes: os desenvolvedores, os analistas de teste, o time de suporte e os donos de produto.

Desenvolvedores despendem tempo lendo e analisando o fluxo e requisitos do software para, então aplicar ajustes solicitados pelo gerente ao código. Uma vez tendo o fluxograma e outros documentos pertinentes do código, é possível agilizar este processo e realizar uma entrega requisitada mais rápida e eficaz. Àqueles que trabalham dando suporte à ferramenta do software também podem se beneficiar de uma documentação mais atualizada do sistema.

Ao realizar testes manuais em um software, um analista de teste deve entender bem o funcionamento esperado para então reportar caso haja algum defeito. Além da parte de interface gráfica, é preciso saber que alterações e ações uma determinada funcionalidade realiza. Com requisitos incompletos ou sem documentação completa, atualizada e confiável os testes ficam passíveis a erros.

Da mesma forma, ao chegar um novo colaborador no time responsável por uma aplicação, é necessário introduzi-lo às ferramentas utilizadas e à aplicação ali desenvolvida. É possível que isso seja feito por algum membro mais antigo do time pelo grande conhecimento do processo, porém, uma documentação de fácil entendimento e acesso possibilita que este responsável por repassar os conhecimentos não esqueça de nenhum detalhe, além de que os tópicos ali aprendidos possam ser recorrentemente consultados.

Diante dos fatos anteriormente descritos, o uso da ferramenta proposta neste trabalho pode ser benéfico para um rápido andamento e manutenção de um projeto já existente.

2 FUNDAMENTAÇÃO TEÓRICA

Os tópicos apresentados neste capítulo tratam dos principais conceitos relacionados ao assunto deste trabalho, sendo estes: manutenibilidade, modelo de requisitos, engenharia reversa e reengenharia. Além disto, é elencado três trabalhos referentes ao uso da engenharia reversa em softwares.

2.1 MANUTENIBILIDADE

A manutenção de um sistema, visto como estágio importante durante sua vida útil, pode acontecer por diversos motivos e em diferentes fases de seu projeto. O termo manutenibilidade deve ser levado em consideração durante a estruturação de um projeto, sendo este a definição da facilidade com que o software poderá receber manutenções ou melhorias. Koscianski e Soares (2007) vão além, e defendem a aplicação de métricas de manutenibilidade, tanto para prever o esforço empregado na manutenção quanto para gerar um histórico durante seu desenvolvimento.

É possível realizar diferentes tipos de manutenção nestes sistemas, com distintas finalidades, conforme explanado por Koscianski e Soares (2007), sendo estas: manutenção adaptativa, na qual acontece a adequação da ferramenta utilizada; manutenção perfectiva, responsável pela adição de novas funcionalidades ao software requisitada pelos chefes de negócio; manutenção corretiva que realiza a correção de erro diante de um mal funcionamento. Este último pode não ser beneficiado diretamente pela engenharia reversa, no entanto o programador pode detectar o erro com maior facilidade se a engenharia reversa for aplicada ao código.

Entretanto, um forte aliado da fase de manutenção é o conhecimento do sistema. Por vezes, isto está confiado apenas à experiência prévia da equipe responsável pelo

trabalho original ou de seus conhecimentos passados a outros membros responsáveis. Esta confiança pode gerar problemas no futuro, uma vez que estes membros podem não estar mais trabalhando neste mesmo projeto ou até mesmo na empresa, não restando assim, ninguém com conhecimento direto do software legado em questão. Por esta razão, possuir uma documentação do sistema é tão importante. Assim como Chikofsky e Cross II (1990) reafirmam durante sua explicação sobre manutenção de software, de que neste contexto, a engenharia reversa é a parte do processo de manutenção que auxilia os envolvidos no mesmo a entenderem o sistema para então realizar as mudanças necessárias.

2.2 MODELOS DE REQUISTOS

A modelagem de requisitos proporciona uma visão ampla do que o software que será desenvolvido, ou que está em andamento, deve realizar em termos de funções e comportamento. De acordo com Koscianski e Soares (2007), a especificação gerada tem como origem conversas com os usuários finais e gerentes de projeto, definições sobre seu funcionamento e detalhes sobre o modelo de negócio aplicado, este processo de coleta de informações é chamado de engenharia de requisitos. A engenharia de requisitos é um papel importante desde o início do projeto, estendendo-se até as fases de manutenção. Koscianski e Soares (2007) destacam ainda que os requisitos devem ser escritos de forma correta e concisa, possibilitando uma fácil compreensão e rápida leitura durante o desenvolvimento do sistema, além de servir como consulta para futuras manutenções.

Conforme descrito por Pressman (2011) o modelo de requisitos é a primeira representação técnica de um sistema e deve alcançar três objetivos primários, são

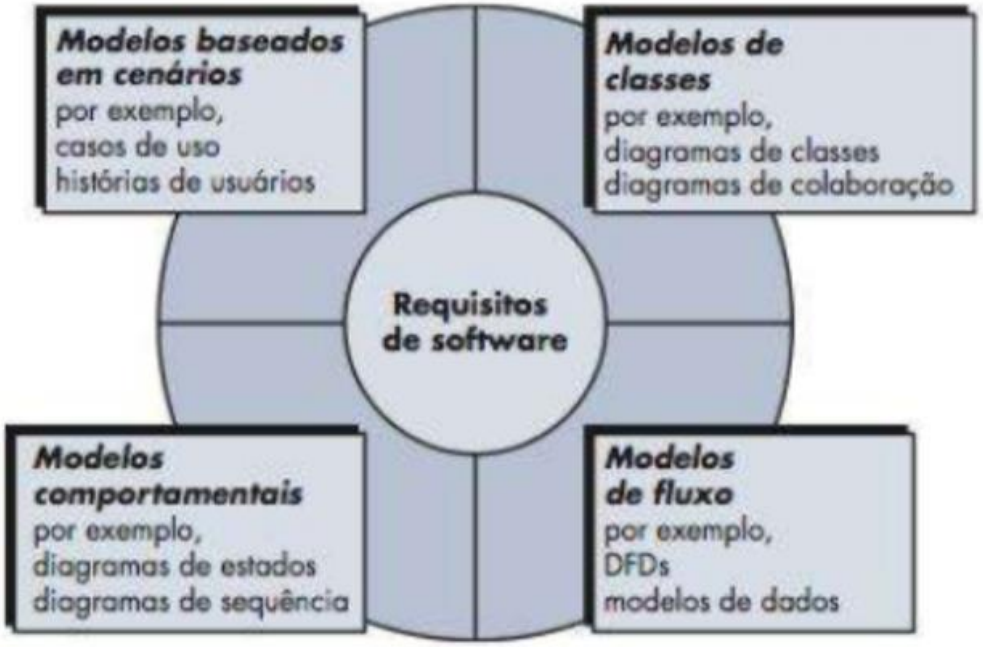
eles: descrever o que o cliente solicita; estabelecer uma base para a criação de um projeto de software; definir um conjunto de requisitos que possa ser validado assim que o software for construído. Para atingir estes objetivos é possível utilizar diversas ferramentas e maneiras de representação das regras do sistema, como listas e diagramas.

Existem maneiras de representar os modelos de requisitos, descrevendo o sistema por diferentes perspectivas, tendo de maneira geral quatro elementos de definição comum, são eles (PRESSMAN, 2011):

- Elementos baseados em cenários: o sistema é descrito do ponto de vista do usuário final;
- Elementos baseados em classes: o sistema é descrito agrupando os comportamentos e atributos comuns;
- Elementos comportamentais: o sistema é descrito conforme o computador irá interpretá-lo;
- Elementos orientados a fluxos: o sistema é descrito na forma de fluxograma.

É possível observar estes modelos mais comuns na Figura 1, contendo alguns exemplos para cada caso. (PRESSMAN, 2011).

Figura 1 – Elementos do modelo de requisitos



Fonte: Pressman (2011, p.155)

Conforme pode ser visto na Figura 1, distintas representações dos requisitos de um software podem ser criadas, encaixando-se nos modelos elencados anteriormente. Pressman (2011) expõe ainda, que estas diferentes maneiras de representação dos modelos de requisitos agregam importante valor aos envolvidos no projeto, podendo ser utilizado como base para o desenvolvimento, consulta durante o planejamento de testes ou análise do ponto de vista da validação do sistema. Nos casos nas quais estas representações do sistema estão defasadas ou são inexistentes, é possível recuperá-las aplicando a engenharia reversa no código-fonte, conforme abordado neste trabalho.

2.3 ENGENHARIA REVERSA

Originado da área de hardware, a engenharia reversa era utilizada ao se desmontar um equipamento na tentativa de descrever a especificação de projeto e fabricação de um produto de empresa concorrente, uma vez que estes documentos originais são de propriedade privada, como menciona Pressman (2011). Na área de engenharia de software, a engenharia reversa tem o mesmo objetivo de compreender o funcionamento e técnicas utilizadas na construção de um sistema, para então, gerar uma descrição em alto nível de abstração do sistema. Sendo desta forma, sua aplicação muito utilizada nos softwares da própria empresa que a aplica, normalmente considerados como softwares legado.

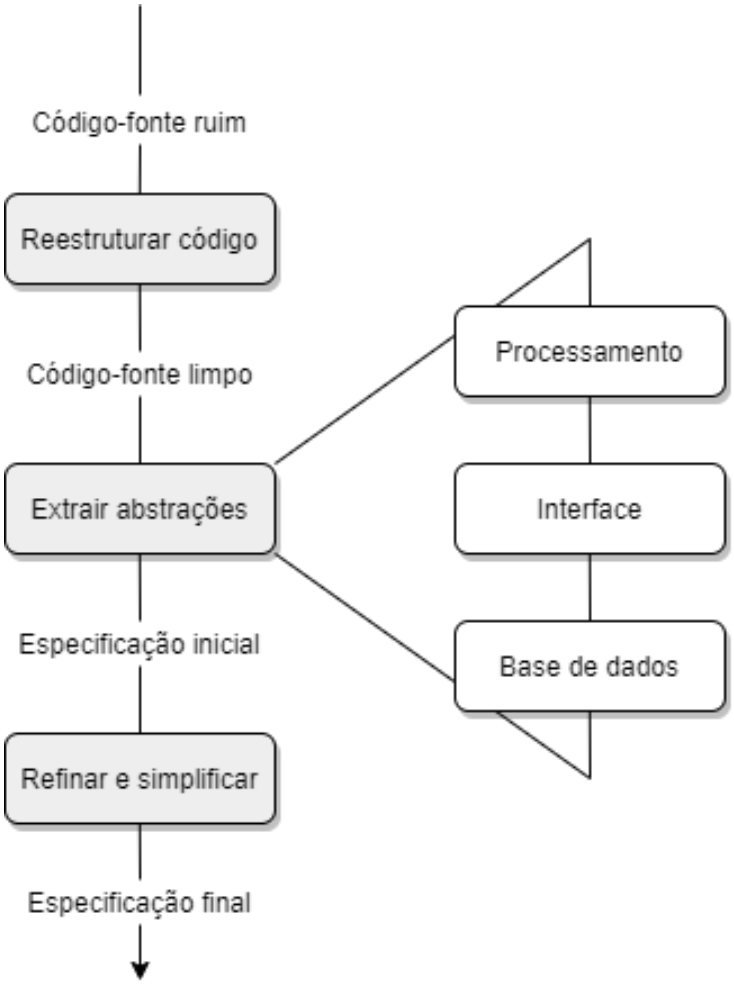
De forma simplificada, a engenharia reversa é definhada como “entendimento do funcionamento interno de um programa”. (PRESSMAN, 2011, p.669). Ou, como dito por Chikofsky e Cross II (1990), engenharia reversa é a análise do sistema com objetivo de identificar os seus componentes e suas relações, além de possibilitar a criação de uma representação em alto nível de abstração do sistema. Chikofsky e Cross II (1990) defendem também, que existem diversas subáreas dentro da engenharia reversa, sendo duas delas muito difundidas: a redocumentação e a recuperação do projeto.

Devido à alta complexidade do sistema a ser analisado utilizando engenharia reversa, sua descrição pode ter diferentes níveis de abstração. Segundo Costa (1997, p.9), “Cada visualização abstrai características próprias da fase do ciclo de vida correspondente à abstração”.

Por vezes, conforme a qualidade da engenharia reversa aplicada, não é possível gerar uma documentação tão detalhada em relação a que deveria ter sido realizada

no início e ao longo do projeto. Por este motivo, os sistemas não são feitos com nenhum tipo de modelo de requisitos para então, posteriormente, aplicar engenharia reversa em seu código. A engenharia reversa vem como uma ferramenta de auxílio, proporcionando informações importantes do programa, geralmente de forma abstrata, conforme descrito por Costa (1997). O processo de engenharia reversa está representado pela Figura 2.

Figura 2 – O processo de engenharia reversa



Fonte: Pressman (2011, p.671)

Como primeiro passo, se necessário, faz-se uma reestruturação do código ruim do sistema, deixando-o mais fácil para ser trabalhado. Como núcleo da engenharia reversa, está a extração de abstrações, no qual os níveis de processamento, interface e banco de dados são analisados e extraídos as principais informações para se desenvolver uma especificação inicial. A partir desta fase, é possível refinar e simplificar esta especificação inicial, tornando sua compreensão mais fácil para utilização dos interessados. (PRESSMAN, 2011).

2.4 ENGENHARIA REVERSA E REENGENAHRIA

Comumente confundido, o termo reengenharia difere de engenharia reversa em alguns aspectos importantes, porém, ambas possuem o mesmo propósito, que é o de auxiliar no bom desempenho de um software legado. (CAGNIN e PENTEADO, 2000).

Com o passar do tempo os sistemas em uso tendem a perder pontos importantes em sua documentação, ou não possuir nenhum tipo de definição utilizada no seu projeto em geral. A engenharia reversa tem como princípio a análise do código-fonte para gerar uma abstração do sistema. Tal descrição serviria como base para a criação deste mesmo sistema em tecnologias mais atualizadas, ou, como defende Cagnin e Penteado (2000), pode ser utilizado como ferramenta de consulta para a manutenção do sistema por ser mais viável financeiramente.

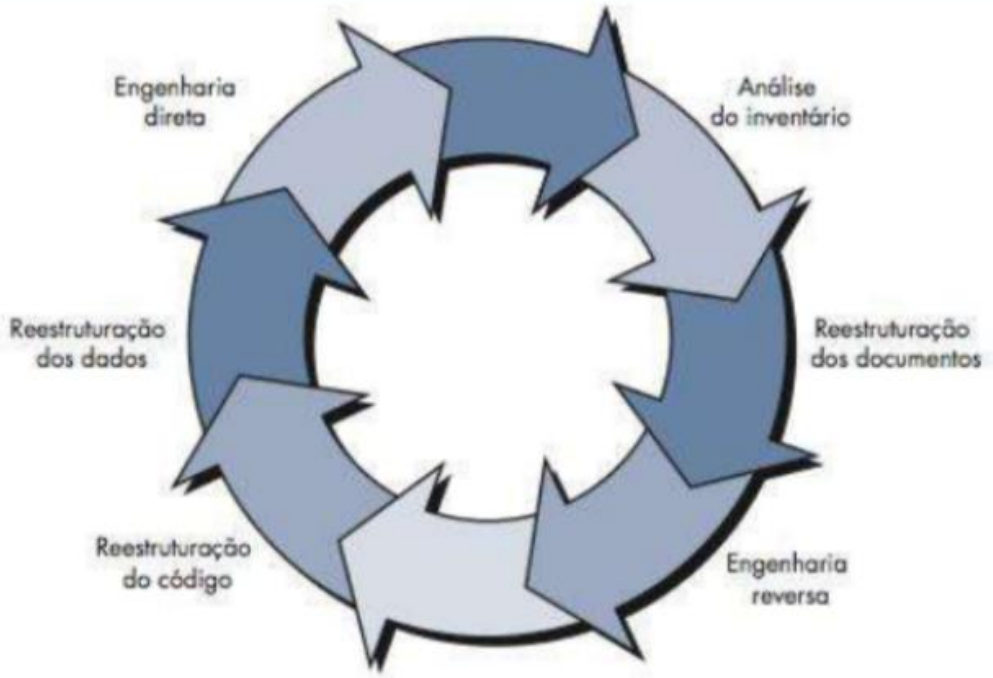
Com grande número de softwares funcionando por tanto tempo, é comum que as tecnologias empregadas no início de seu desenvolvimento estejam defasadas e manutenções não sejam mais eficientes ou vantajosas financeiramente. Percebe-se ao longo da vida útil de um sistema que manutenções não atendem mais às necessidades de negócio impostas pelo mercado, “de fato, não é raro uma

organização de software despender de 60% a 70% de todos os recursos com manutenção de software”. (PRESSMAN, 2011, p.663). É possível afirmar que em certos casos apenas a aplicação de engenharia reversa não é suficiente, sendo necessário utilizar, também a reengenharia.

A reengenharia é utilizada nestes casos para recriar o sistema a partir de sua documentação, renovando assim, o tempo de vida do sistema com: linguagens e ferramentas atualizadas. Apesar de possuir um custo alto e demandar tempo com a familiarização das novas tecnologias utilizadas, como aponta Cagnin e Penteado (2000), pode ser uma boa opção de ferramenta para refatoração do código.

Em resumo, a engenharia reversa analisa o código-fonte para obter e documentar o sistema de uma forma mais abstrata, enquanto a reengenharia analisa o documento de projeto para planejar a restruturação do sistema ou criar um novo do mesmo. Desta forma, observa-se que a engenharia reversa pode ser utilizada como parte da reengenharia na falta de uma documentação do sistema. A Figura 3 mostra as etapas do ciclo da reengenharia, contendo a engenharia reversa. (PRESSMAN, 2011).

Figura 3 – Ciclo do processo de reengenharia



Fonte: Pressman (2011, p.668)

Descrito por Pressman (2011), a reengenharia de sistemas é um processo cíclico, com suas etapas em constante execução. Sem um ponto de início e fim bem delimitado, a reengenharia pode partir das etapas de análise de inventário e reestruturação dos documentos, buscando um melhor entendimento dos elementos existentes e funcionamento do sistema. A engenharia reversa vem logo em seguida, criando uma representação do programa ao resgatar informações faltantes diretamente do código-fonte. Após o conhecimento completo do funcionamento do sistema, é possível realizar a reestruturação do código e reestruturação dos dados, com o objetivo de torná-los mais atualizados, reescrevendo o código com outra linguagem de programação e reformulando a arquitetura de dados. Por último, mas não menos importante, a engenharia direta é realizada, em casos em que as

informações do projeto são recuperadas, utilizando-as para alterar ou reconstituir o sistema.

2.5 TRABALHOS RELACIONADOS

Conforme apresentado por Magalhães (2008), tem-se estudado sobre a usabilidade da engenharia reversa nos softwares e suas vantagens, tanto em sua aplicação isolada, quanto o uso da mesma como parte do processo de reengenharia. A seguir, descreve-se trabalhos relacionados à aplicação da engenharia reversa em softwares e suas devidas características.

2.5.1 Uma proposta de Engenharia Reversa Baseada em Testes de Software

Em seu trabalho Ávila (2013) propôs uma abordagem da engenharia reversa através dos resultados de testes funcionais de software. Segundo Ávila (2013), realizar engenharia reversa utilizando o código-fonte e a experiência dos desenvolvedores do sistema, tal resultado pode apresentar erros oriundos da fonte. Teste funcional de software, por sua vez, possui uma visão de cliente, considerando apenas valores de entrada e saída esperados do sistema.

Conforme explicado por Ávila (2013), seu trabalho utiliza duas técnicas de abordagem, sendo elas a documental e a de estudo de casos. A avaliação quanto a possibilidade de criação de casos de uso com engenharia reversa é feita comparando o documento gerado pelo seu trabalho, baseado em testes do software, com o documento original de casos de uso do software.

Após análise dos casos de teste escolhidos foi identificado elementos para a construção dos casos de uso, como por exemplo, identificação de atores, interfaces, fluxos e requisitos específicos do sistema. Com os casos de uso gerados utilizando

conceitos de engenharia reversa, Ávila (2013) pôde fazer a comparação com os casos de uso originais do sistema. Esta avaliação foi feita seguindo alguns critérios de fluxo e subfluxos esperados e identificados, regras de negócio atendidos e cobertura alcançada.

Ávila (2013) constatou que para que os resultados sejam precisos, é necessário que os casos de teste utilizados sejam concisos e possuam uma cobertura ampla dos requisitos do sistema. Em tempo, considerou os casos de teste como boa fonte para utilização na engenharia reversa e definiu a proposta como aceitável “Conclui-se que a proposta é aceitável, considerando que as notas de cobertura total foram 3 e 4, tendo como referência o valor máximo 5”. (Ávila, 2013, p.6).

Levando em consideração o trabalho apresentado e resultados obtidos por Ávila (2013), é possível observar algumas similaridades com o trabalho aqui desenvolvido. Assim como o autor utilizou documentações já existentes, para realizar as comparações e validação do sistema desenvolvido, este trabalho também usa documentações pré-existentes dos softwares analisados para confirmar sua eficácia. Desta forma, é possível validar a ferramenta desenvolvida e a aplicação de prova de conceito.

2.5.2 Aplicação da Engenharia Reversa e Reengenharia de Software no Desenvolvimento de Plugins para a Ferramenta Oryx

A monografia de Sousa e Leite (2012) teve como objetivo desenvolver *plugins* para a ferramenta Oryx através da aplicação de engenharia reversa e reengenharia. Sousa e Leite (2012) consideram que existe pouca documentação referente à infraestrutura do software de modelagem Oryx, por este motivo, a engenharia reversa

foi aplicada como um primeiro passo à reengenharia e criação dos *plugins* propostos inicialmente.

Em suma, “A ferramenta Oryx é um *framework* acadêmico de código aberto inicialmente desenvolvido para oferecer a modelagem gráfica de processos de negócio.”. (SOUSA e LEITE, 2012, p.9). Sendo este um código aberto, é possível consultar e utilizar o código-fonte livremente. Este trabalho utiliza um método baseado em um metaprocesso, dividido em quatro subprocessos: recuperar, especificar, reprojetar e reimplementar; sendo a engenharia reversa aplicada no primeiro subprocesso.

Ainda dentro da etapa de engenharia reversa, foi dividida em três fases: instalação da ferramenta Oryx e obtenção do código-fonte; identificação da arquitetura e definições necessárias para gerar o *plugin*; análise do ambiente de programação do Oryx e formação de seus principais elementos e relacionamentos. Com tais informações adquiridas, Sousa e Leite (2012) comentaram que o panorama das necessidades alcançadas se tornou mais clara.

Independentemente do objetivo final do trabalho apresentado por Sousa e Leite (2012) ser a aplicação de reengenharia, a engenharia reversa foi imprescindível no início do projeto. Foi possível perceber que a organização e divisão de etapas básicas dentro deste subprocesso, foram úteis para resgatar a infraestrutura do software. Tais etapas na engenharia reversa também foram utilizadas no decorrer deste trabalho, sendo elas: obtenção, identificação da arquitetura e formação de principais elementos do código-fonte.

2.5.3 An Overall Process Based on Fusion to Reverse Engineer Legacy Code

O artigo escrito por Penteado, Germano e Masiero (1996) descreve o uso e análise do método de engenharia reversa Fusion, e apresentou o Fusion/RE partindo deste método. O novo processo Fusion/RE apresentado teve como melhorias propostas a edição e simulação de diagramas, além de fornecer métricas de conversão.

O método Fusion tem fácil ligação entre as fases executadas e é composto por basicamente três etapas: análise, *design* e implementação. A fase de análise usa expressões regulares para especificar como o sistema se comunica. Após cada operação de análise de entradas e saídas, um esquema textual é modelado. No *design* cada operação é refinada em gráficos interativos para cada classe ou objeto identificado. Já na fase de implementação, estes gráficos gerados são utilizados para montar a nova interface do sistema.

Penteado, Germano e Masiero (1996) explicam que o objetivo do processo proposto, Fusion/RE, é utilizar os princípios de Fusion em três etapas: revitalizar a arquitetura do sistema, criar um modelo de análise e generalizá-lo em um modelo de análise abstrato. Uma quarta etapa pode ser adicionada para conectar os modelos de análise obtidos. Para realizar a modelagem, o sistema é divido em partes como entradas, saídas, dados gravados de forma temporária ou definitiva. A estrutura de dados é extraída e modelada conforme seus objetos de classe.

Após aplicar o modelo Fusion/RE em código legados, Penteado, Germano e Masiero (1996) puderam tirar conclusões em relação ao seu uso. Alguns problemas foram identificados no sistema analisado, reafirmando assim sua importância para futuras manutenções.

Após a confirmação da importância do uso de engenharia reversa em códigos legados, foi possível fundar-se no artigo de Penteado, Germano e Masiero (1996) para a realização deste trabalho. Da mesma maneira que foi apresentada por eles, este trabalho tem o propósito de realizar a análise do sistema e gerar seus resultados de forma abstrata e generalizada.

3 METODOLOGIA

Para atingir o objetivo de solução da problemática apresentada ao longo deste trabalho, utilizou-se o método de pesquisa de natureza prática, com o intuito de desenvolver uma solução aplicável à sociedade. Para isso, desenvolveu-se um software genérico como uma possível solução à falta de documentação de sistemas legados, utilizando conceitos de engenharia reversa como base de conhecimento. Ocasionalmente, foi necessário realizar uma pesquisa do tipo exploratória, para melhor entendimento dos conceitos utilizados e melhor estruturação dos resultados propostos. Conforme dito por Moresi (2003), a pesquisa exploratória é o primeiro passo quando não se tem total conhecimento sobre o assunto, e “Por sua natureza de sondagem, não comporta hipóteses que, todavia, poderão surgir durante ou ao final da pesquisa.” (MORESI, 2003, p.9).

Como forma de coleta de dados para realizar a aplicação do sistema desenvolvido, buscou-se coletar códigos-fonte de sistemas abertos, disponíveis livremente na internet. Todos os códigos-fonte utilizados, possuem algum tipo de documentação já existente, para fins de comparação de resultados posteriores. Seguiu-se os seguintes passos como critério na escolha dos códigos-fonte: aberto; escrito em linguagem de programação Java; possuir algum tipo de documentação; ter sido desenvolvido há mais de cinco anos. Apesar do último critério não ser suficiente para caracterizar um sistema como legado, escolheu-se este fator como forma de simulação durante este trabalho.

Como parte da análise de resultados da solução proposta desenvolvida, foi utilizado o procedimento de pesquisa experimental, em que a solução proposta é aplicada e testada em situações reais. Moresi (2003) afirma que, na pesquisa

experimental é possível observar um fenômeno dada a aplicação de uma condição, conforme realizado nesta fase deste trabalho. Após a coleta dos dados para análise, o sistema desenvolvido foi aplicado nestes códigos-fonte e seus resultados validados. A partir desta validação em um sistema já existente, foi possível deduzir sua eficácia e precisão em cima dos resultados obtidos.

4 DESENVOLVIMENTO DO SOFTWARE

Neste capítulo, as etapas de desenvolvimento do software são detalhadas. As ferramentas utilizadas estão descritas, os requisitos funcionais e não funcionais são elencados entre outras condições necessárias para o entendimento do seu desenvolvimento. Subsequentemente, o desenvolvimento do código é apresentado, detalhando seus passos realizados, bem como a forma de utilização do sistema em outros códigos-fonte. Por fim, é relatado o tipo de código-fonte em que este sistema pode ser aplicado e seus resultados esperados.

Este software tem acesso ao sistema já existente a ser analisado, no qual aqui chamou-se de código-fonte, extraindo dele as informações necessárias para o entendimento do seu fluxo e funcionamento.

Após a extração dos dados, o software desenvolvido organizou as informações obtidas e assim gerou os documentos explicativos. Tais documentos apresentam-se na forma de texto descritivo, para melhor entendimento do público-alvo, mas sempre buscando um nível alto de abstração das informações.

4.1 FERRAMENTAS UTILIZADAS

A seguir, é elencado as ferramentas utilizadas durante este trabalho. Caso desejado, é possível utilizar outras ferramentas semelhantes, desde que seja mantido características como sua linguagem de programação, para um correto funcionamento do código.

Para o desenvolvimento deste sistema, em relação ao editor de código escolheu-se o Visual Studio Code, desenvolvido pela Microsoft, pela sua robustez e familiaridade com seu funcionamento. Para fins de execução do sistema, usou-se ou

o terminal do Windows, também conhecido como *Prompt* de Comando, ou o terminal integrado no editor de texto Visual Studio Code. Como forma de hospedagem e versionamento de código, contou-se com a plataforma GitHub, em repositório de conta pessoal privada. (VISUAL STUDIO CODE, 2021).

Quanto as linguagens de programação foram utilizadas JavaScript - linguagem de programação interpretada estruturada capaz de implementar funcionalidades complexas - e Node.js como seu ambiente de execução para o desenvolvimento da lógica do *BackEnd*. (NODE.JS, 2021). O Node.js fica encarregado por realizar a execução da aplicação do lado do servidor e, posteriormente, mostrar os resultados obtido diretamente no terminal de comando ou salvá-los em arquivo de texto.

Para a parte visual do sistema, chamada de *FrontEnd*, optou-se por montar uma página web, utilizando às linguagens HTML e CSS. Enquanto a linguagem de marcação de hipertextos HTML fica responsável por montar a parte estrutural da página web, a linguagem de estilos CSS é encarregada da parte estética dos resultados obtidos. (MDN WEB DOCS, 2021). A utilização das tecnologias escolhidas permite que se visualize resultados tanto por meio de navegadores web, de forma gráfica, quanto por meio de arquivos, de forma textual.

4.2 ANÁLISE DE REQUISITOS

A seguir serão apresentados os requisitos funcionais e não funcionais do sistema, para melhor definição de suas funções.

4.2.1 Requisitos Funcionais

De acordo com Sommerville (2011), requisitos funcionais retratam os serviços que o sistema deve fornecer, bem como suas reações a entradas específicas e comportamento. Dito isto, os requisitos funcionais deste trabalho são:

- O usuário deve ser capaz de aplicar o sistema em qualquer código-fonte escrito na linguagem Java de sua preferência;
- O usuário deve definir como entrada o código-fonte escolhido, adicionando-o na pasta indica para isto;
- O usuário deve executar o sistema para que o mesmo processe os resultados;
- O sistema deve ter total acesso ao código-fonte a ser analisado;
- O sistema deve abrir e ler todos os arquivos disponibilizados pelo usuário;
- Para cada arquivo lido, uma ação deve ser tomada, conforme seu conteúdo;
- O sistema deve extrair informações relevantes e organizá-las para a construção da documentação como resultado;
- O sistema deve fornecer os resultados obtidos na forma estilizada através de uma página web e na forma textual através de arquivos de texto;
- O sistema deve permitir que se exporte os resultados em formato PDF.

4.2.2 Requisitos Não Funcionais

Requisitos não funcionais, por sua vez, descrevem possíveis restrições do sistema como um todo para seu funcionamento, não se aplicando á funcionalidades

visíveis ao usuário final, conforme explicado por Sommerville (2011). Neste trabalho, os seguintes requisitos não funcionais foram adotados:

- A lógica do sistema deve ser desenvolvida utilizando linguagem de programação JavaScript;
- O sistema deve executar de forma rápida e eficaz;
- O sistema deve ser escrito de forma genérica e aplicável em qualquer código-fonte escrito na linguagem de programação Java;
- O sistema não deve alterar nenhum código-fonte do código a ser analisado;

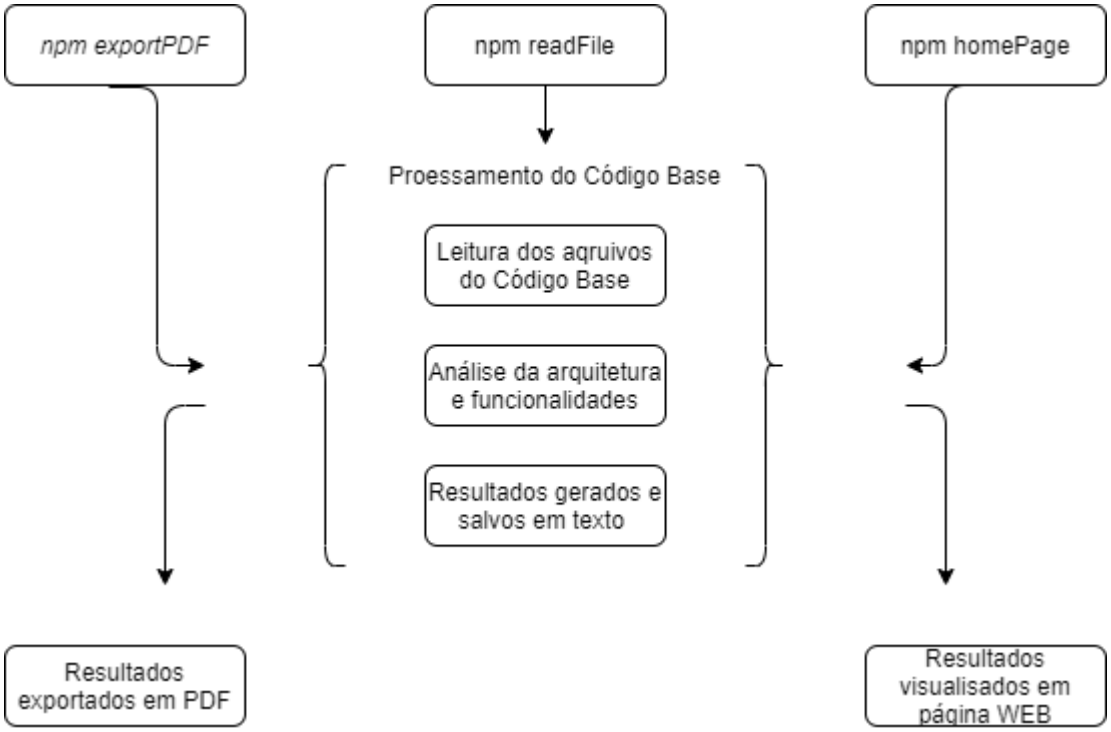
4.3 DESENVOLVIMENTO DO CÓDIGO

A seguir, é apresentado os elementos principais do desenvolvimento deste trabalho, tais como, uma visão geral do sistema, o diagrama de casos de uso, a estrutura do projeto e a principal lógica aplicada.

4.3.1 Visão Geral

Como propósito principal, este software faz a análise de um código-fonte para então extrair as suas principais funções. É possível identificar o fluxo destas atividades, em uma forma de alto nível, conforme mostra a Figura 4.

Figura 4 – Funcionalidades do Software.



Fonte: Elaborado pela Autora (2021)

Como mostra a Figura 4, o módulo Processamento do Código Base é o núcleo do processamento deste software, no qual é realizada a leitura dos arquivos do código-fonte, a análise da arquitetura e das funcionalidades é feita e em seguida os resultados são gerados e salvos em arquivo de texto.

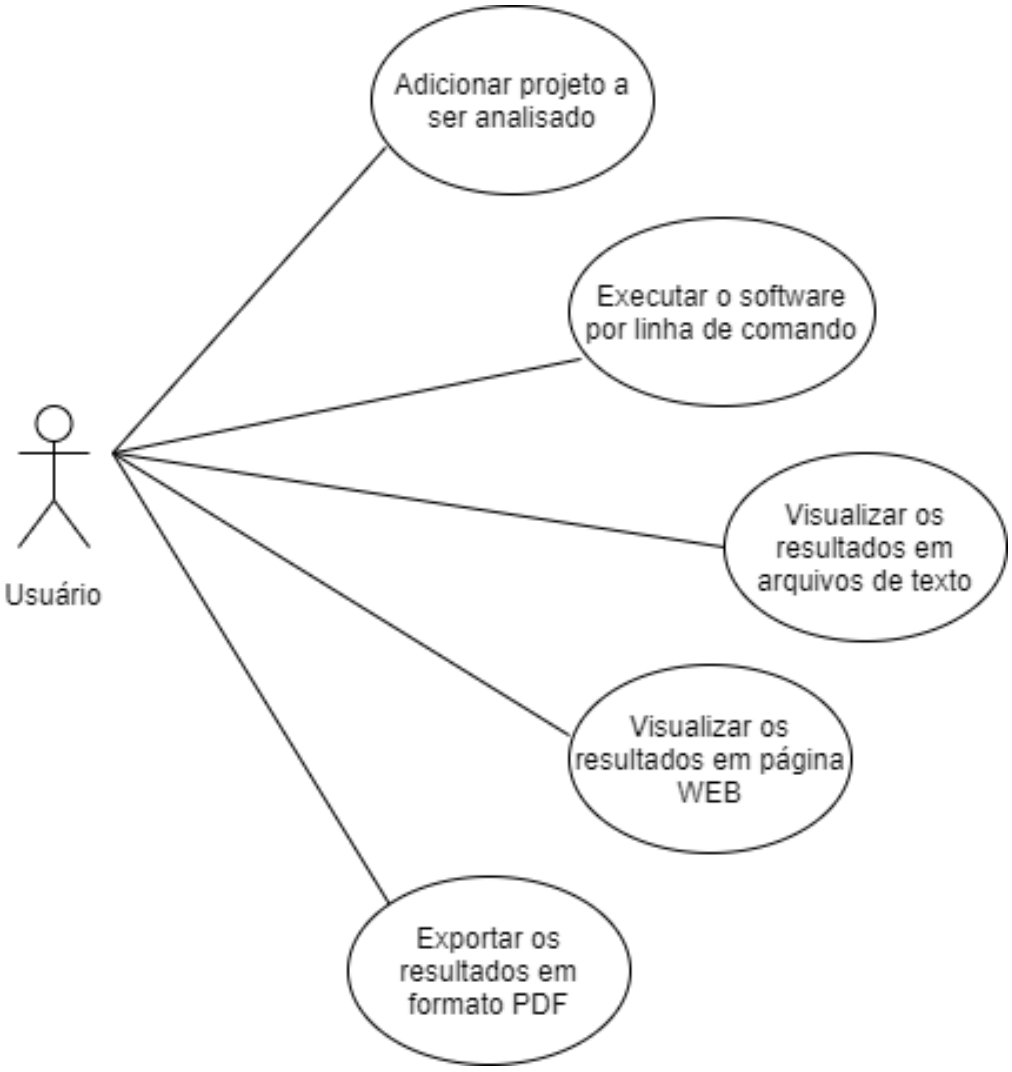
O fluxograma da Figura 4 mostra ainda, que conforme o comando executado pelo usuário, um tipo diferente de saída dos resultados é gerado. Para efetuar apenas o que contempla o módulo de Processamento do Código Base, é preciso executar o comando *npm readFile*, desta forma os resultados são obtidos e salvos em arquivo de texto. Caso o comando *npm exportPDF* seja executado, ele chama o módulo Processamento do Código Base para, então, exportar os resultados em formato PDF. Também há a possibilidade de visualizar os resultados em uma página web, para tal,

o comando *npm homePage* deve ser executado, no qual chama o módulo principal e logo em seguida, abre uma instancia web com as informações obtidas.

4.3.2 Diagrama de Casos de Uso

O diagrama de casos de uso serve como um modelo dos elementos possíveis de serem observados no sistema ou externos a ele. Na Figura 5, é mostrado o diagrama de casos de uso do sistema, do ponto de vista do usuário.

Figura 5 – Diagrama de casos de uso do sistema.



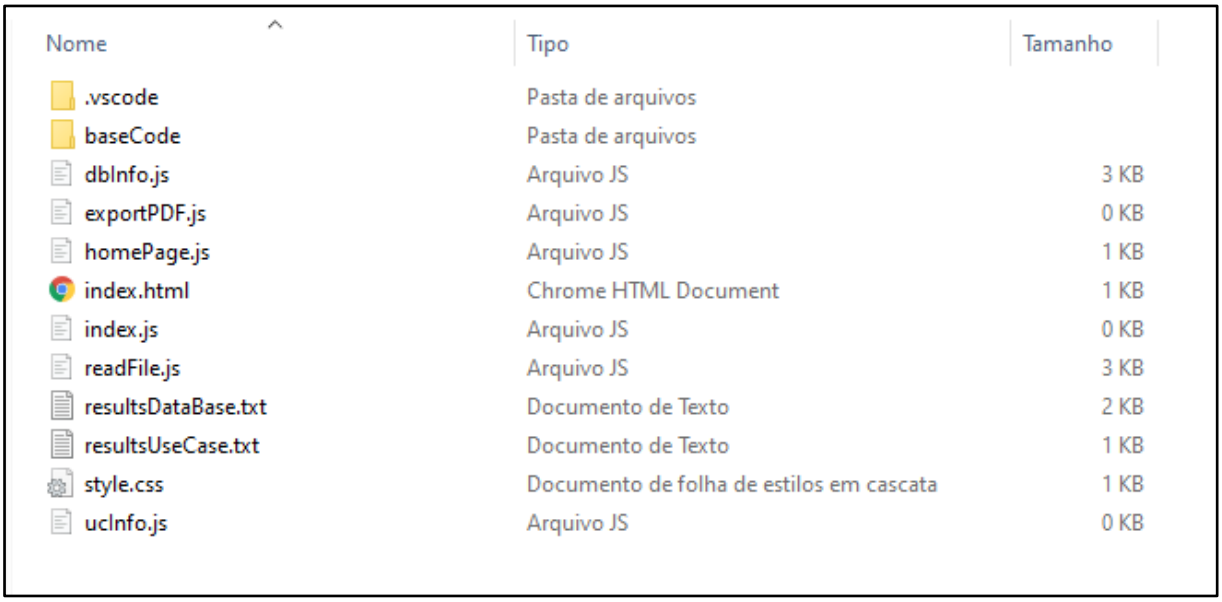
Fonte: Elaborado pela Autora (2021)

O usuário deve adicionar o código-fonte que deseja ser avaliado, na pasta denominada *baseCode*, para que assim o software possa reconhecê-lo e utilizá-lo para análises. Uma vez que o código foi adicionado no projeto, o usuário pode executar o software a qualquer momento. Assim que executado e processado as informações, é possível visualizar os resultados de duas formas, sendo elas: em arquivo de texto ou em página web. Além disto, o usuário pode também, caso deseje, exportar os resultados obtidos para um arquivo em formato PDF.

4.3.3 Estrutura

O software desenvolvido tem uma estrutura acessível e direta, com os elementos principais necessários para a aplicação da análise no código-fonte. A Figura 6 mostra em detalhes a organização dos diretórios criados para este trabalho, em seguida, é explicado a função de cada um no mesmo.

Figura 6 – Estrutura do Software.



Fonte: Elaborado pela Autora (2021)

Como parte dos requisitos, o usuário deve adicionar o código-fonte a ser aplicado à engenharia reversa dentro do projeto. Para tal, foi criado a pasta *baseCode* para que ele possa adicionar os elementos a serem trabalhados. Os resultados gerados após sua execução, podem ser vistos diretamente nos arquivos *resultsDataBase.txt* e *resultsUseCase.txt*

4.3.4 Algoritmo de Manipulação de *String*

Para o processamento principal do código-fonte foi utilizado manipulação de *strings*, identificando assim os principais elementos descritos neste código. Para detectar padrões e tipos de funções, foi necessário comparar trechos das *strings* com palavras chaves, como por exemplo: *admin, cliente* e *table*. Na Figura 7 é possível visualizar o trecho em que o arquivo é comparado com palavras chaves, para então, chamar as devidas funções conforme o tipo de dados ali encontrado.

Figura 7 – Comparação com *Strings.*

```
function processFile (data) {
    if (data.toLowerCase().includes('create table'.toLowerCase())) {
        console.log('This file contains DB info');
        dbInfo.extractDBInfo(data);
    }else if (data.toLowerCase().includes('role'.toLocaleLowerCase())) {
        console.log('This file contains UserCase info');
        ucInfo.extractUCInfo(data);
    } else {
        console.log('This file has no relevant information');
    }
};
```

Fonte: Elaborado pela Autora (2021)

Uma vez identificada o tipo de informação capaz de ser extrair do trecho sendo analisado, utilizou-se o método do *JavaScript* chamado de *split(),* que permite separar

uma *string* lida por operadores ou caracteres pré-definidos. Para uma extração mais detalhada de elementos da *string*, utilizou-se o objeto *RegExp()*. A expressão regular, ou *RegExp(),* é um objeto que descreve um padrão de caracteres, usado para buscas ou substituição de textos. Um trecho do código desenvolvido, contendo o método *split()* e o objeto *RegExp(),* podem ser vistos na Figura 8. Este trecho corresponde à extração de informações relacionadas ao banco de dados do código-fonte a ser analisado.

Figura 8 – Utilização do Objeto *RegExp* e Método S*plit*.

```
function columnNames(data) {
    const stringExtractor = extract(['\.',',']);
    const stuffIneed = stringExtractor(data);

    //correct first item extracted
    let firstItem = stuffIneed[0].split('(').slice(2).join('(');
    stuffIneed[0] = firstItem;

    //properly extract last item
    let lastItem = data.split(',').pop().split(')').shift();
    stuffIneed[stuffIneed.length] = lastItem;

    for (let i = 0; i < stuffIneed.length; i++) {
        fs.appendFile('resultsDataBase.txt',   ' ' + stuffIneed[i] + '\n', (err) => {
            if(err) throw err;
        });
    }

    return stuffIneed;
};

function extract([beg, end]) {
    const matcher = new RegExp(`${beg}(.*?)${end}`,'gm');
    const normalise = (str) => str.slice(beg.length*+0,end.length*-1);
    return function(str) {
        return str.match(matcher).map(normalise);
    }
};
```

Fonte: Elaborado pela Autora (2021)

Na função de extração de informações relacionados ao banco de dados, acontece uma verificação dentro do arquivo aberto, procurando-se alguma *string*

chamada ‘*create table*’. Caso afirmativo, o código percorre o arquivo e separa as palavras por indicadores como virgulas e parênteses. Desta forma, os nomes de tabela e as colunas correspondentes são retirados e salvos no arquivo *resultsDataBase.txt.*

Já na função de extração de informações de casos de uso, o código desenvolvido, depois de ter acesso e abrir o arquivo a ser analisado, verifica se tal arquivo possui a *string* ou parâmetro ‘*role*’, onde possivelmente indica os atores do sistema. Uma vez encontrado, e se não há nenhum outro parâmetro como ‘role’, considera-se que cada chamada para outras classes com extensão *.jsp*, indica uma ação que este usuário pode realizar. Extrai-se estas classes então, eliminado a extensão, para montar a lista de ações do usuário. É importante salientar que estas classes precisam ter um nome com significado, para que seu resultado faça sentido.

Com tais ferramentas, foi possível identificar e extrair informações detalhadamente dos arquivos do código-fonte. A partir desta extração de informações, foi possível reuni-las e alimentar os resultados, inicialmente nos arquivos de texto pré-determinados para tal função.

4.4 UTILIZAÇÃO

Este software visa sua utilização pelos membros do time responsável pelo sistema em análise, por este motivo, espera-se algum conhecimento básico de instalação e utilização das tecnologias aqui mencionadas. É preciso primeiramente garantir a disponibilidade das ferramentas utilizadas. Como primeiro passo, recomenda-se que faça a instalação do editor de texto proposto ou outro de sua preferência, além da instalação do executador Node.js, necessário para rodar os

comandos definidos no JavaScript. Após, é necessário adquirir o código aqui desenvolvido.

Uma vez que a instalação das ferramentas necessárias e a aquisição do código desenvolvido foram feitas, define-se o código-fonte a ser analisado. É possível utilizar para fins de teste um dos códigos disponíveis, como Code With C (2014a), assim como foi feito na análise realizada neste trabalho, ou, pode-se utilizar outro código desde que seja escrito em linguagem de programação Java. Tal código deve ser copiado para dentro do diretório denominado *baseCode*, que se encontra abaixo do diretório do projeto principal *RevEngDoc*. Desta forma, o sistema desenvolvido será capaz de identificar o código a ser analisado.

Para realizar a execução, é necessário abrir o terminal do sistema operacional, ou o terminal integrado do Visual Studio Code, digitar o comando *node readFile* e apertar a tecla Enter. Em seguida, o sistema será executado e os resultados obtidos serão gravados nos arquivos de texto *resultsDataBase.txt* e *resultsUseCase.txt*. É possível também, executar o comando *node homePage*, onde o sistema será executado e uma instancia web é inicializada, contendo os resultados. Como terceira opção, pode-se executar o comando *node exportPDF* para extrair os resultados em arquivo formato PDF.

4.5 CÓDIGOS BASE

Como forma de validação do software desenvolvido, utilizou-se códigos-fonte já existentes, disponíveis na internet de forma aberta. Os códigos-fonte escolhidos possuem algum tipo de documentação existente, como por exemplo: diagramas de casos de uso; fluxogramas, ou; texto descritivo de suas funcionalidades. Estes

documentos serviram como base para a validação da eficácia da ferramenta desenvolvida, o que possibilitou a realização da comparação entre o documento existente e o gerado pelo software proposto.

Conforme requisitos mencionados anteriormente, os códigos-fonte escolhidos têm como critério ser de acesso livre, que tenham sido escritos em linguagem de programação Java, possuir documentação e terem sido desenvolvidos há um certo tempo, neste caso definiu-se cinco anos para efeitos de simulação. A linguagem de programação Java foi escolhida por motivos de conhecimento prévio na tecnologia, agilizando assim o processo de análise durante o desenvolvimento do trabalho.

Juntamente com os códigos-fonte escolhidos, é possível fazer a aquisição de sua documentação através do endereço disponibilizado neste trabalho. A seguir, na Tabela 1, é apresentado os códigos-fonte escolhidos e listado se possuem diagrama de casos de uso ou informações de entidade relacionamento em sua documentação.

Tabela 1 – Códigos-fonte Escolhidos.

Código-fonte	Possui diagrama entidade relacionamento.	Possui diagrama de casos de uso.
CODE WITH C (2014a)	X	X
CODE WITH C (2014b)	X	
CODE WITH C (2014c)	X	X
CODE WITH C (2015)		X

Fonte: Elaborado pela Autora (2021)

5 COLETA E ANÁLISE DE DADOS

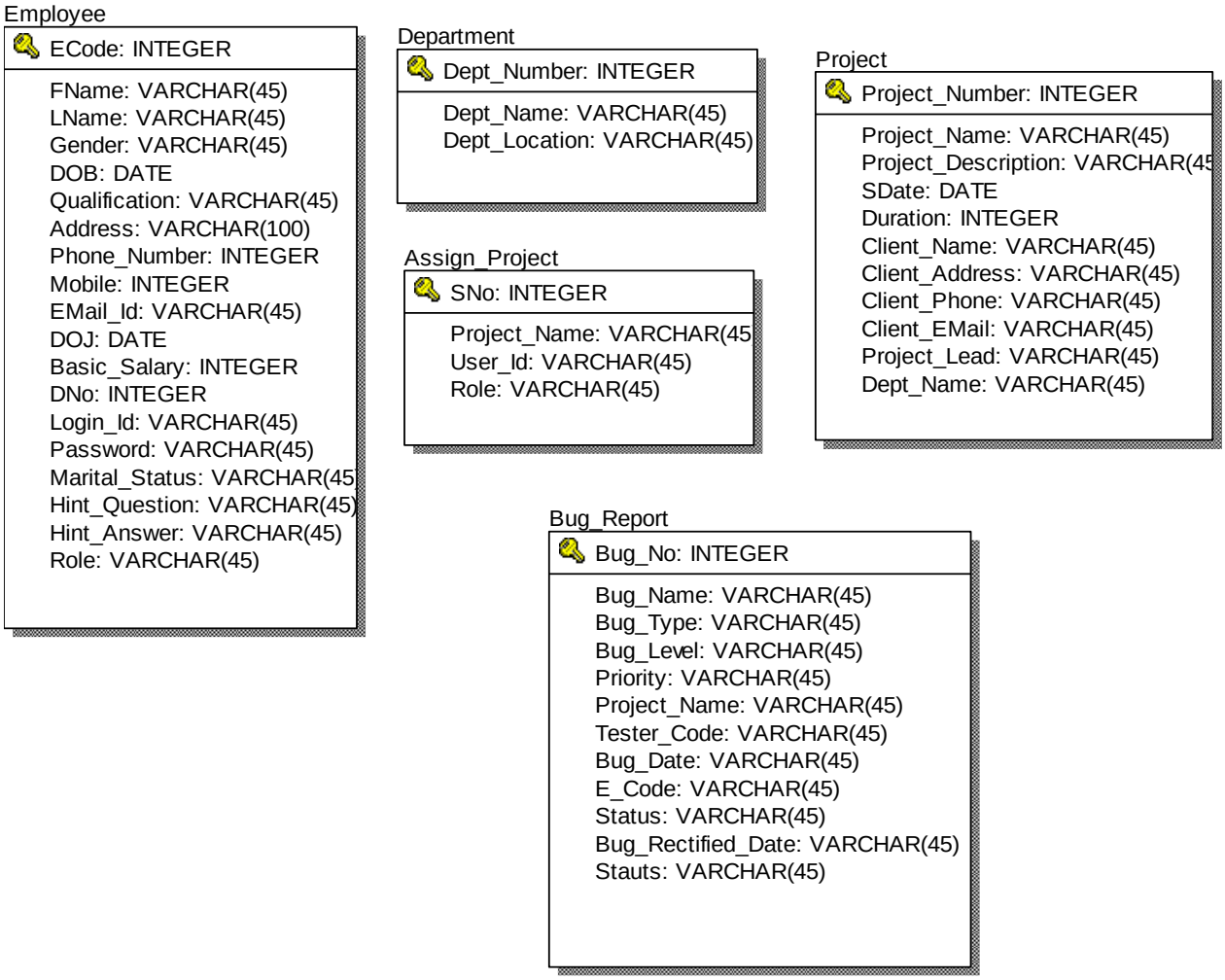
Após o desenvolvimento do software proposto realizou-se a aplicação do mesmo em códigos-fonte, para fins de teste de sua funcionalidade, possibilitando a extração de dados de comparação, conforme mostrados nos tópicos a seguir. Como o software desenvolvido não produz seus resultados em forma de gráficos nesta primeira fase, a comparação foi feita de forma visual, ao ler e verificar o seu conteúdo gerado com a documentação existente.

5.1 RESULTADOS RELACIONADOS AO BANCO DE DADOS

Foi extraído resultados em relação ao banco de dados de três códigos-fonte utilizados neste trabalho, devido a sua disponibilidade de informações. Nos três casos, os códigos possuem diagramas de Entidade e Relacionamento, ilustrando a estrutura lógica do banco de dados. O resultado gerado com informações de banco de dados está listado da seguinte forma: a palavra ‘Table name’ seguida do nome da tabela, listando abaixo dela o nome das colunas. Os resultados das tabelas estão separados por uma linha em branco.

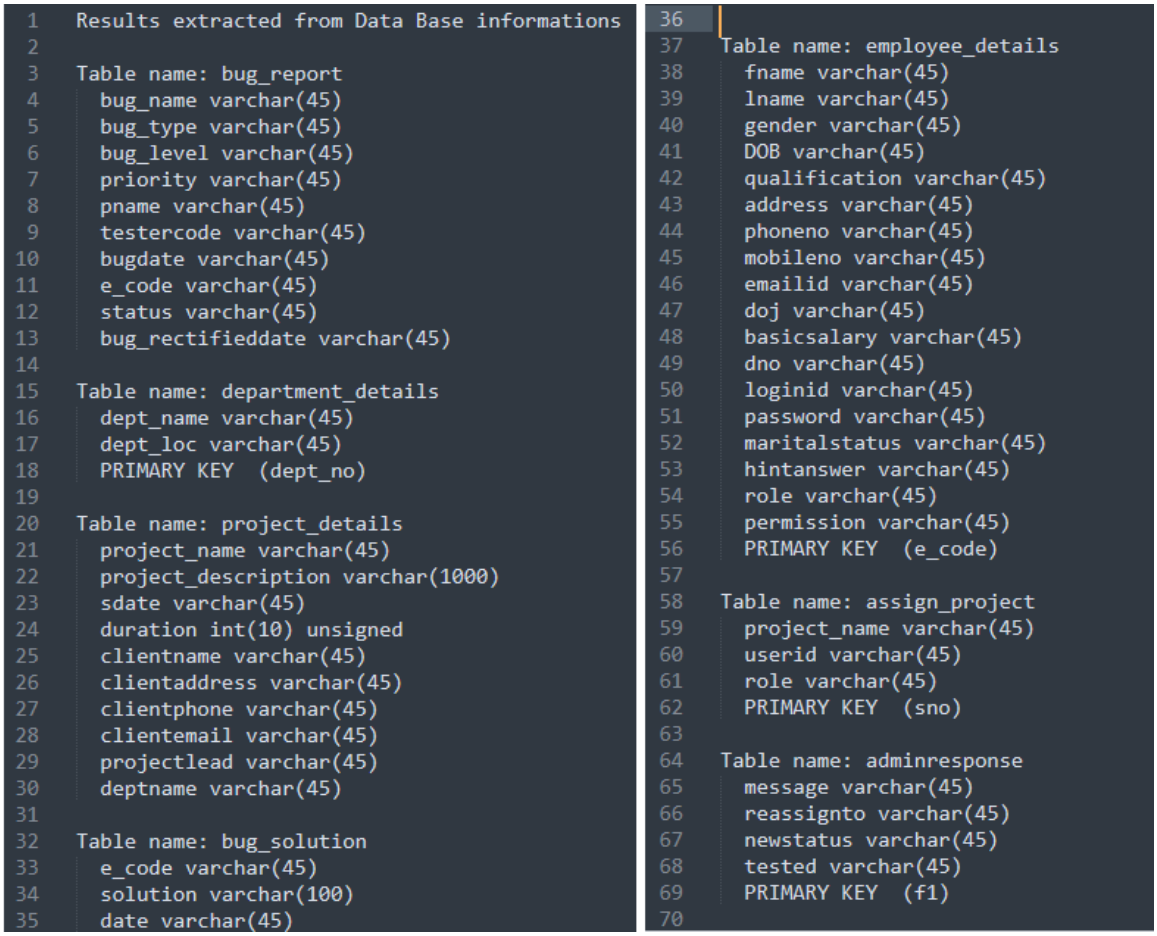
A seguir, no primeiro teste de validação, a Figura 9 mostra o diagrama entidade e relacionamento disponível em Code With C (2014a), e a Figura 10 mostra o resultado obtido da extração de informações relacionados ao banco de dados do mesmo trabalho.

Figura 9 – Entidade e Relacionamento Teste 1



Fonte: Code With C (2014a, p.30)

Figura 10 – Resultado Banco de Dados Teste 1



```
Results extracted from Data Base informations

Table name: bug_report
  bug_name varchar(45)
  bug_type varchar(45)
  bug_level varchar(45)
  priority varchar(45)
  pname varchar(45)
  testercode varchar(45)
  bugdate varchar(45)
  e_code varchar(45)
  status varchar(45)
  bug_rectifieddate varchar(45)

Table name: department_details
  dept_name varchar(45)
  dept_loc varchar(45)
  PRIMARY KEY  (dept_no)

Table name: project_details
  project_name varchar(45)
  project_description varchar(1000)
  sdate varchar(45)
  duration int(10) unsigned
  clientname varchar(45)
  clientaddress varchar(45)
  clientphone varchar(45)
  clientemail varchar(45)
  projectlead varchar(45)
  deptname varchar(45)

Table name: bug_solution
  e_code varchar(45)
  solution varchar(100)
  date varchar(45)

Table name: employee_details
  fname varchar(45)
  lname varchar(45)
  gender varchar(45)
  DOB varchar(45)
  qualification varchar(45)
  address varchar(45)
  phoneno varchar(45)
  mobileno varchar(45)
  emailid varchar(45)
  doj varchar(45)
  basicsalary varchar(45)
  dno varchar(45)
  loginid varchar(45)
  password varchar(45)
  maritalstatus varchar(45)
  hintanswer varchar(45)
  role varchar(45)
  permission varchar(45)
  PRIMARY KEY  (e_code)

Table name: assign_project
  project_name varchar(45)
  userid varchar(45)
  role varchar(45)
  PRIMARY KEY  (sno)

Table name: adminresponse
  message varchar(45)
  reassignto varchar(45)
  newstatus varchar(45)
  tested varchar(45)
  PRIMARY KEY  (f1)
```

Fonte: Elaborado pela Autora (2021)

Observa-se que, as cinco tabelas mostradas no diagrama de entidade e relacionamento do código-fonte Code With C (2014a) (Figura 9) estão, também listados nos resultados. Além disto, contém as informações de tipo de dados e tamanho do dado entre parênteses. Porém, é possível notar que duas tabelas que foram extraídas não estão catalogadas no diagrama original, sendo estas: *bug_solution* e *adminresponse.* Isto leva a concluir que, ou o diagrama não foi descrito por completo, ou que, alterações foram feitas após sua criação.

Já no segundo teste realizado, no código-fonte Code With C (2014b), não há um diagrama de entidade e relacionamento disponível, apenas uma tabela descritiva com

informações suficientes para realizar a comparação. É possível visualizar tal tabela na Figura 11 extraída de sua documentação, e na Figura 12 as informações extraídas do mesmo.

Figura 11 – Informações de Banco de Dados Teste 2

1) Table Name: Register

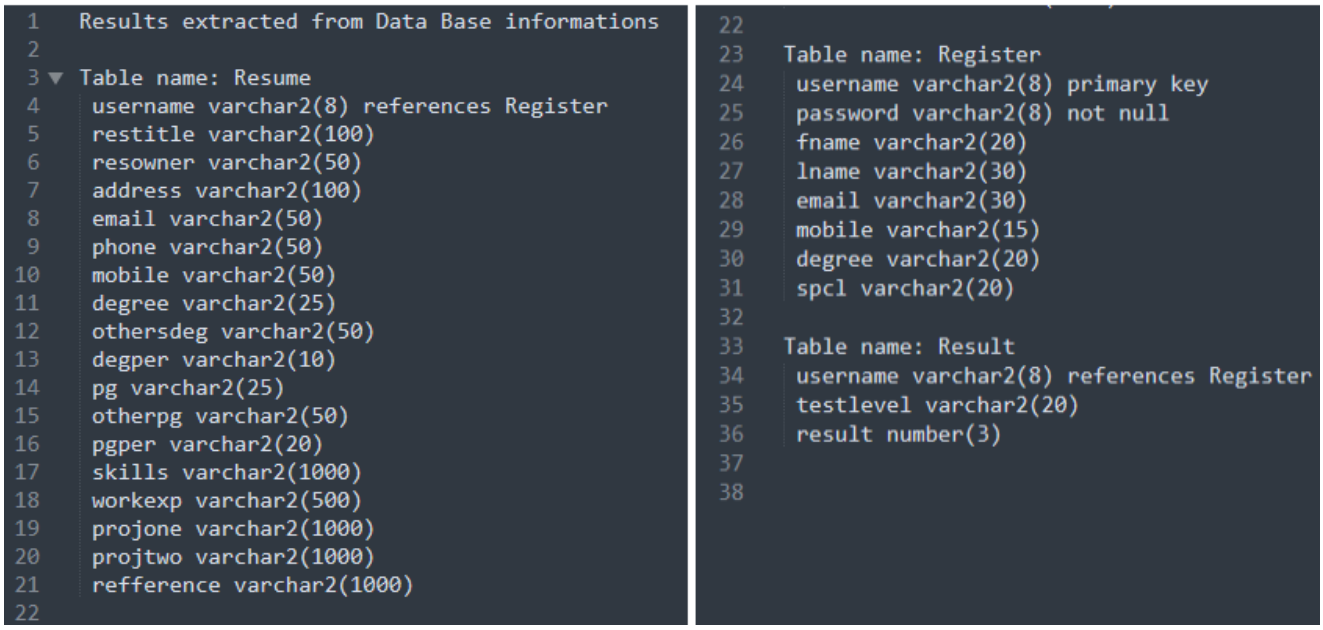
Name	Null?	Type
USERNAME	NOT NULL	VARCHAR2(8)
PASSWORD	NOT NULL	VARCHAR2(8)
FNAME		VARCHAR2(20)
LNAME		VARCHAR2(30)
EMAIL		VARCHAR2(30)
MOBILE		VARCHAR2(15)
DEGREE		VARCHAR2(20)
SPCL		VARCHAR2(20)

2) Table Name: Resume

Name	Null?	Type
USERNAME	NOT NULL	VARCHAR2(8)
RESTITLE		VARCHAR2(100)
RESOWNER		VARCHAR2(50)
ADDRESS		VARCHAR2(100)
EMAIL		VARCHAR2(50)
PHONE		VARCHAR2(50)
MOBILE		VARCHAR2(50)
DEGREE		VARCHAR2(25)
OTHERSDEG		VARCHAR2(50)
DEGPER		VARCHAR2(10)
PG		VARCHAR2(25)
OTHERPG		VARCHAR2(50)
PGPER		VARCHAR2(20)
SKILLS		VARCHAR2(1000)
WORKEXP		VARCHAR2(500)
PROJONE		VARCHAR2(1000)
PROJTWO		VARCHAR2(1000)
REFFERENCE		VARCHAR2(1000)

Fonte: Code With C (2014b, p.86)

Figura 12 – Resultado Banco de Dados Teste 2



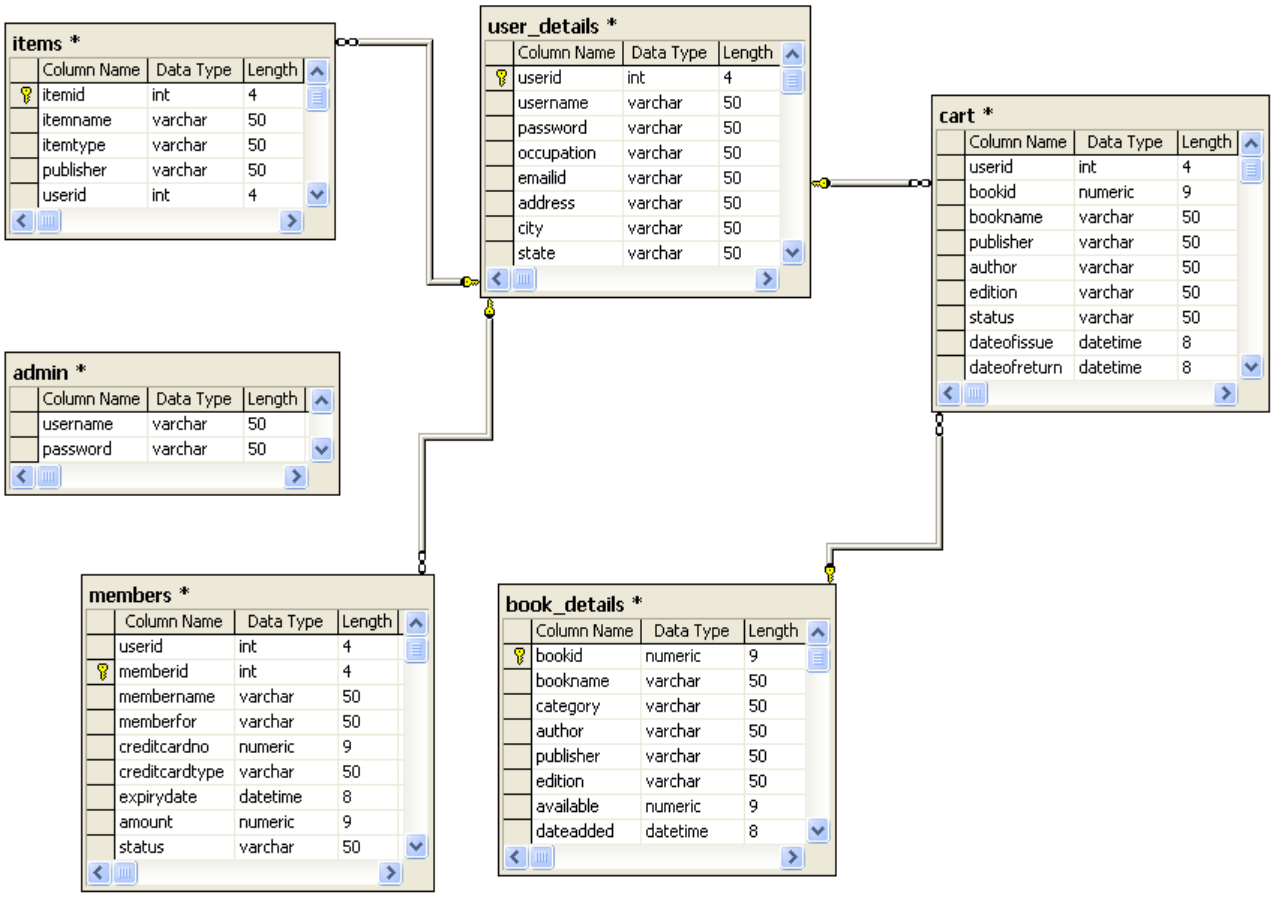
```
Results extracted from Data Base informations

Table name: Resume
 username varchar2(8) references Register
 restitle varchar2(100)
 resowner varchar2(50)
 address varchar2(100)
 email varchar2(50)
 phone varchar2(50)
 mobile varchar2(50)
 degree varchar2(25)
 othersdeg varchar2(50)
 degper varchar2(10)
 pg varchar2(25)
 otherpg varchar2(50)
 pgper varchar2(20)
 skills varchar2(1000)
 workexp varchar2(500)
 projone varchar2(1000)
 projtwo varchar2(1000)
 refference varchar2(1000)

Table name: Register
 username varchar2(8) primary key
 password varchar2(8) not null
 fname varchar2(20)
 lname varchar2(30)
 email varchar2(30)
 mobile varchar2(15)
 degree varchar2(20)
 spcl varchar2(20)

Table name: Result
 username varchar2(8) references Register
 testlevel varchar2(20)
 result number(3)
```

Fonte: Elaborado pela Autora (2021)

Neste caso, no resultado extraído, pode-se identificar as duas tabelas do banco de dados definidas na documentação original, além de informações extras de tipo e tamanho de dados. Ainda assim, é possível notar uma terceira tabela inexistente na documentação disponibilizada, da mesma maneira que o teste anterior.

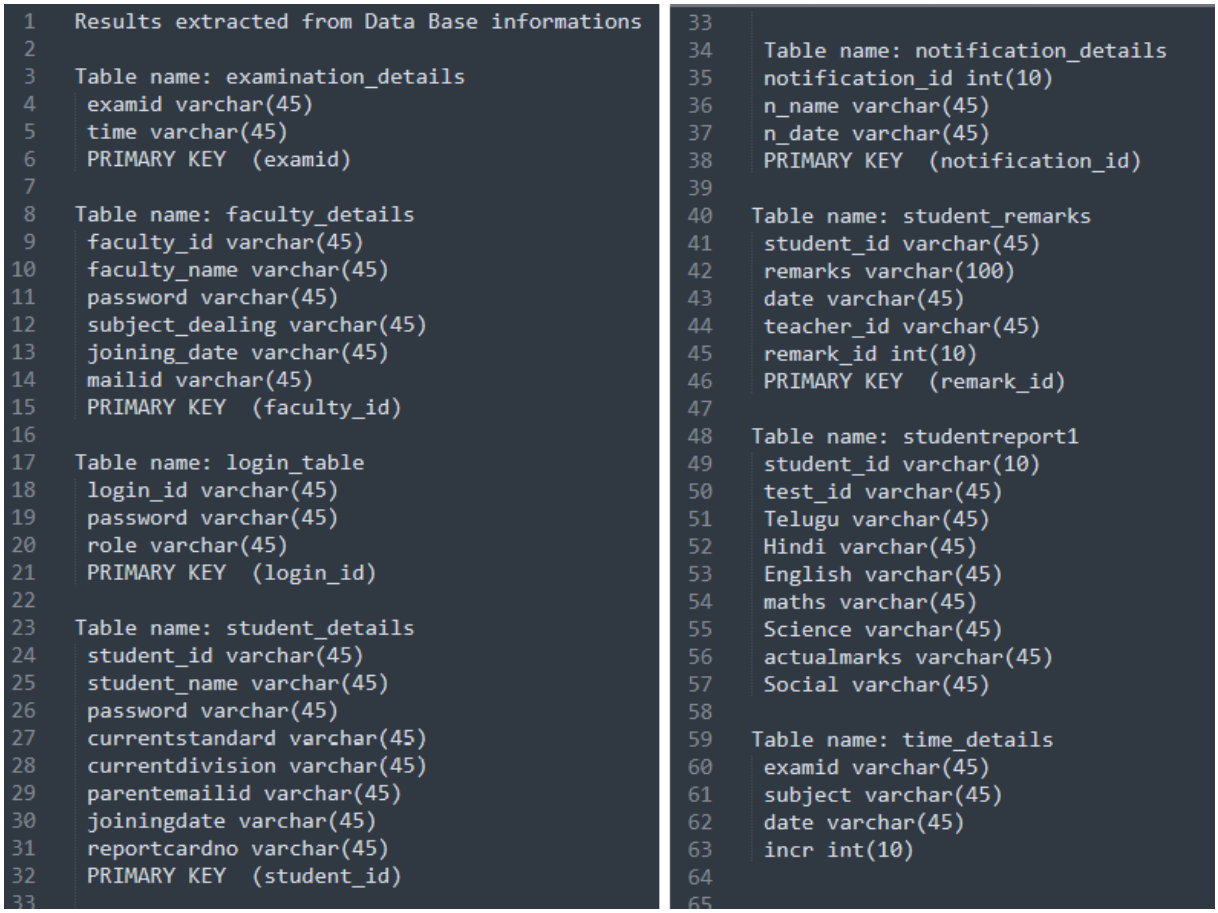
Como terceiro e último código-fonte testado em relação às informações de banco de dados, está o Code With C (2014d). Neste caso, não foi possível tirar um resultado satisfatório. O diagrama de entidade e relacionamento disponibilizado não condiz com o código do mesmo projeto. A Figura 13 possuí este diagrama, enquanto a Figura 14 possui o resultado obtido para este código-fonte.

Figura 13 – Entidade e Relacionamento Teste 3



Fonte: Code With C (2014d, p.23)

Figura 14 – Resultado Banco de Dados Teste 3



```
Results extracted from Data Base informations

Table name: examination_details
 examid varchar(45)
 time varchar(45)
 PRIMARY KEY  (examid)

Table name: faculty_details
 faculty_id varchar(45)
 faculty_name varchar(45)
 password varchar(45)
 subject_dealing varchar(45)
 joining_date varchar(45)
 mailid varchar(45)
 PRIMARY KEY  (faculty_id)

Table name: login_table
 login_id varchar(45)
 password varchar(45)
 role varchar(45)
 PRIMARY KEY  (login_id)

Table name: student_details
 student_id varchar(45)
 student_name varchar(45)
 password varchar(45)
 currentstandard varchar(45)
 currentdivision varchar(45)
 parentemailid varchar(45)
 joiningdate varchar(45)
 reportcardno varchar(45)
 PRIMARY KEY  (student_id)

Table name: notification_details
 notification_id int(10)
 n_name varchar(45)
 n_date varchar(45)
 PRIMARY KEY  (notification_id)

Table name: student_remarks
 student_id varchar(45)
 remarks varchar(100)
 date varchar(45)
 teacher_id varchar(45)
 remark_id int(10)
 PRIMARY KEY  (remark_id)

Table name: studentreport1
 student_id varchar(10)
 test_id varchar(45)
 Telugu varchar(45)
 Hindi varchar(45)
 English varchar(45)
 maths varchar(45)
 Science varchar(45)
 actualmarks varchar(45)
 Social varchar(45)

Table name: time_details
 examid varchar(45)
 subject varchar(45)
 date varchar(45)
 incr int(10)
```

Fonte: Elaborado pela Autora (2021)

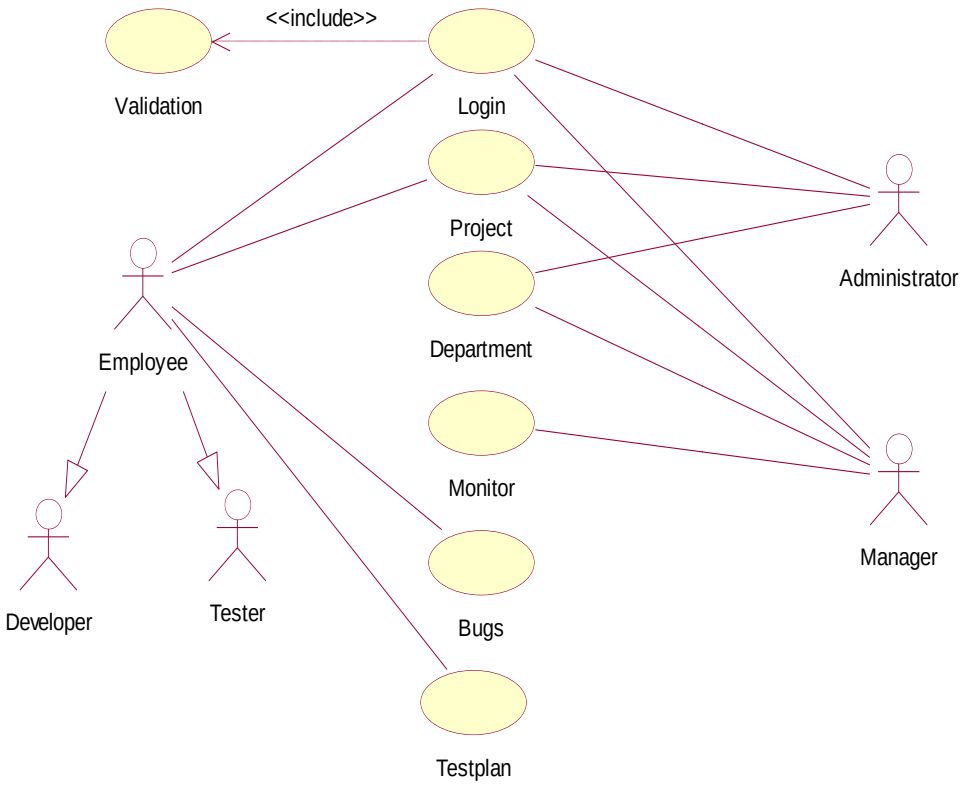
Conforme a comparação visual feita entre a Figura 13 e a Figura 14, os nomes de tabela e colunas são totalmente diferentes. Enquanto a Figura 13, com o diagrama original, tem tabelas com títulos como *book_detail* e *cart*, o resultado extraído e mostrado na Figura 14 possui tabelas com nomes como *student_details* e *examination_details.* Uma verificação manual e detalhada foi feita no código-fonte, para confirmar as diferenças e, atestou-se que o código se referia às mesmas palavras obtidas nos resultados. Tal discrepância leva a crer que o diagrama disponibilizado junto à documentação não faz parte deste código-fonte.

5.2 RESULTADOS RELACIONADOS AOS CASOS DE USO

A aplicação do software desenvolvido nos códigos-fonte permite também, a extração de informações capazes de construir um diagrama de casos de uso. Diagramas de casos de uso representam as funcionalidades do sistema do ponto de vista de seus usuários, chamados de atores, e são representados de forma gráfica. Entretanto, as comparações dos diagramas de casos de uso com as informações obtidas no software aqui desenvolvido, foram feitas de forma visual, pois os resultados do mesmo estão na forma de textos.

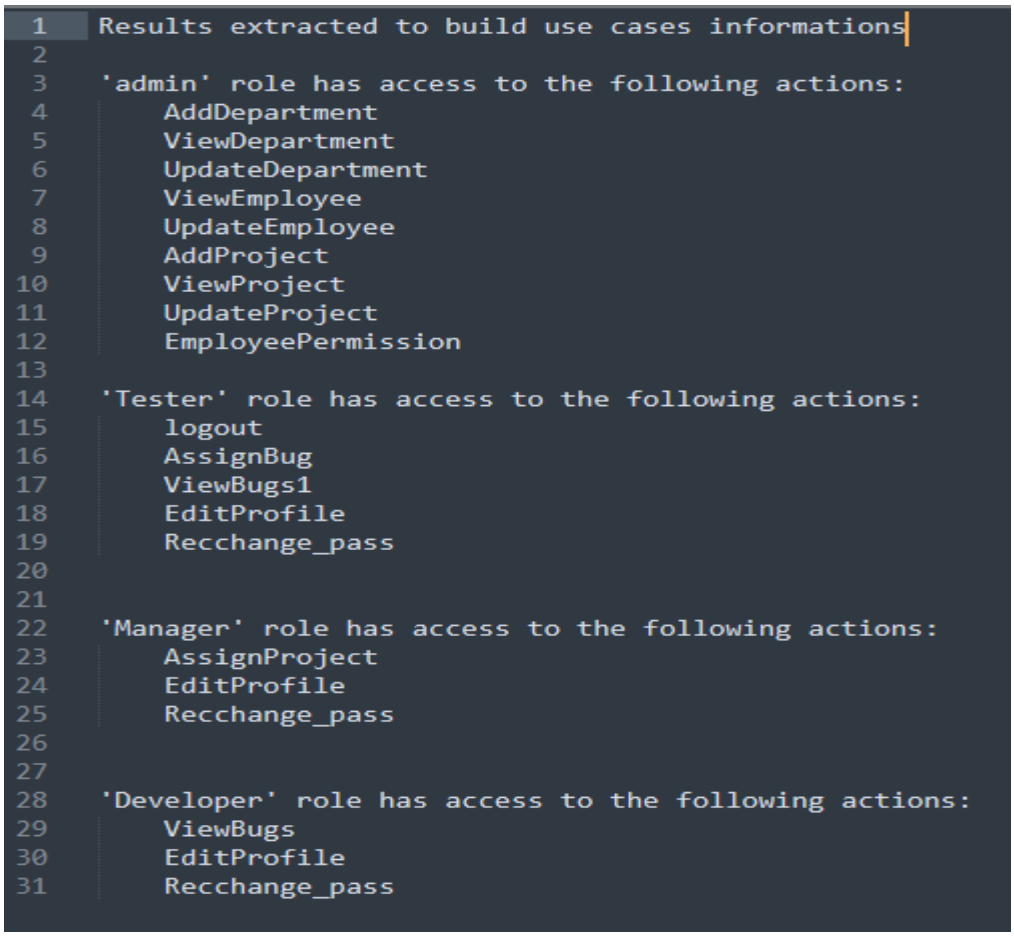
Como primeiro teste, está a análise do software Code With C (2014a). A Figura 15 representa o diagrama de casos de uso disponibilizado, e na Figura 16 estão os resultados obtidos.

Figura 15 – Casos de Uso Teste 1



Fonte: Code With C (2014a, p.18)

Figura 16 – Resultado Casos de Uso Teste 1



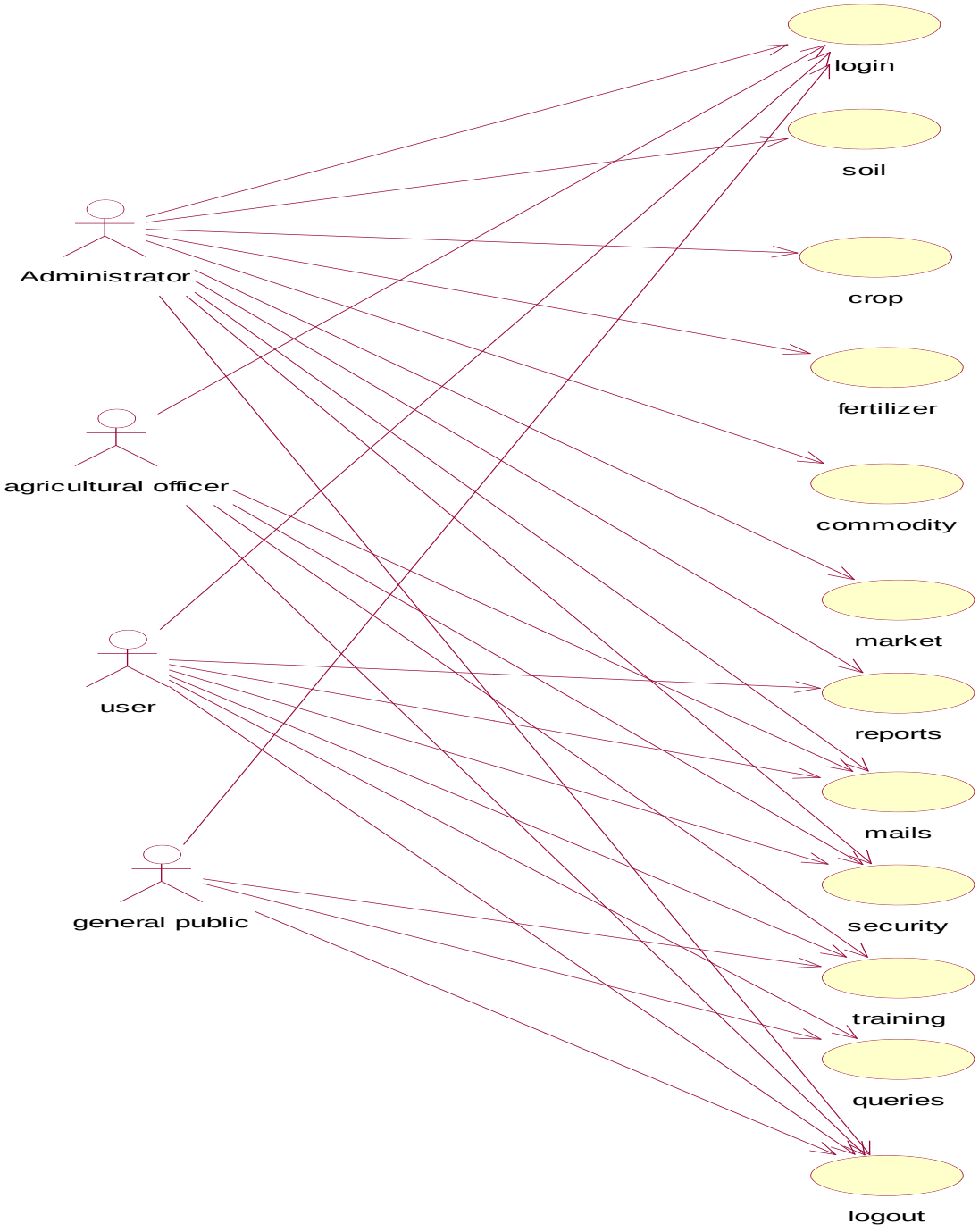
```
Results extracted to build use cases informations

'admin' role has access to the following actions:
    AddDepartment
    ViewDepartment
    UpdateDepartment
    ViewEmployee
    UpdateEmployee
    AddProject
    ViewProject
    UpdateProject
    EmployeePermission

'Tester' role has access to the following actions:
    logout
    AssignBug
    ViewBugs1
    EditProfile
    Recchange_pass


'Manager' role has access to the following actions:
    AssignProject
    EditProfile
    Recchange_pass


'Developer' role has access to the following actions:
    ViewBugs
    EditProfile
    Recchange_pass
```

Fonte: Elaborado pela Autora (2021)

Conforme visto na Figura 15, o autor considerou os atores *Developer* e *Tester* como um ator chamado de E*mployee*, enquanto no resultado obtido aparecem os dois primeiros apenas. Em relação às ações encontradas, é necessária uma interpretação maior dos resultados, pois o que foi extraído do código-fonte possui um nível de detalhamento maior. Exemplificando, na Figura 15 aparece *Department* para o ator *Administrator*, de outro lado, aparecem *AddDepartment, ViewDepartment* e *UpdateDepartment* para o mesmo ator nos resultados. Além disto, percebe-se que estas ações exemplificadas deveriam estar relacionadas também ao ator *Manager* segundo o documento original, não aparecendo na figura com os resultados.

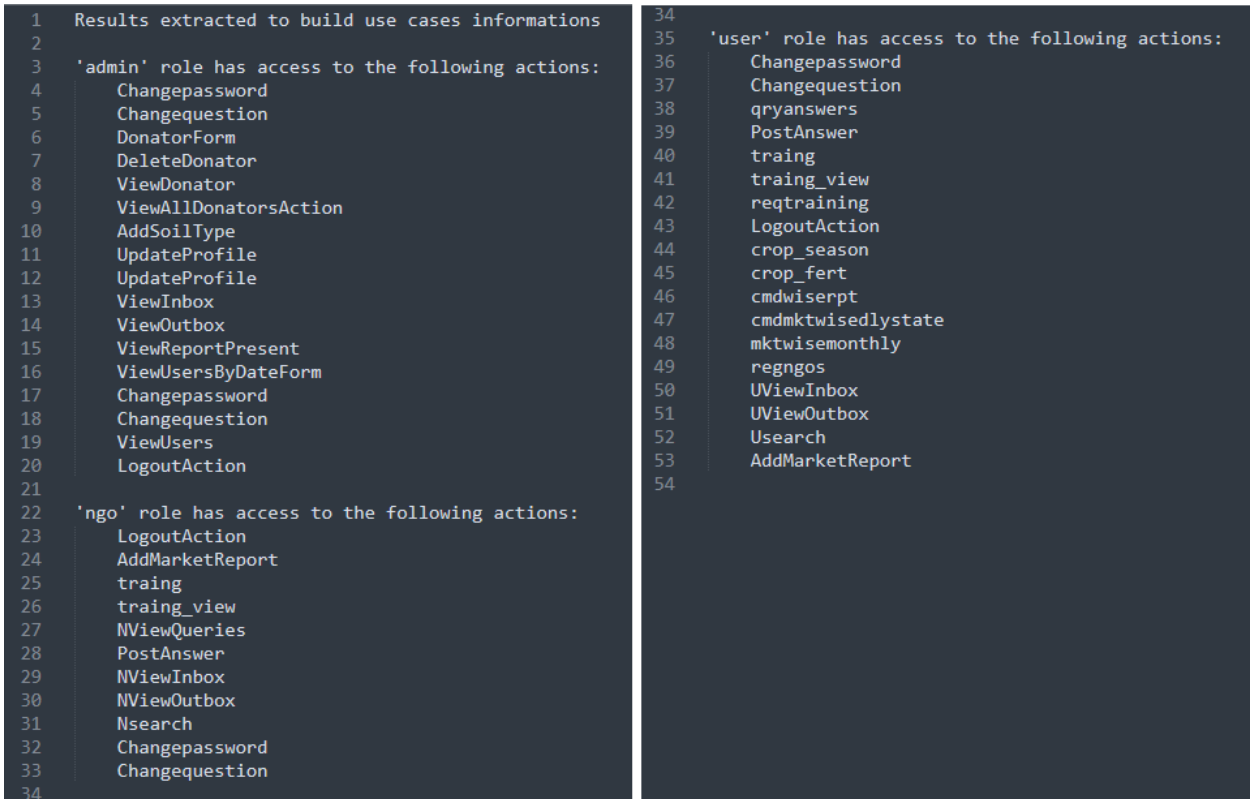
Para dar prosseguimento aos testes, foi analisado o software Code With C (2015). Na Figura 17 encontra-se o diagrama de casos de uso originais deste sistema, e na Figura 18 estão os resultados obtidos em forma de lista.

Figura 17 – Casos de Uso Teste 2



Fonte: Code With C (2015, p.23)

Figura 18 – Resultado Casos de Uso Teste 2



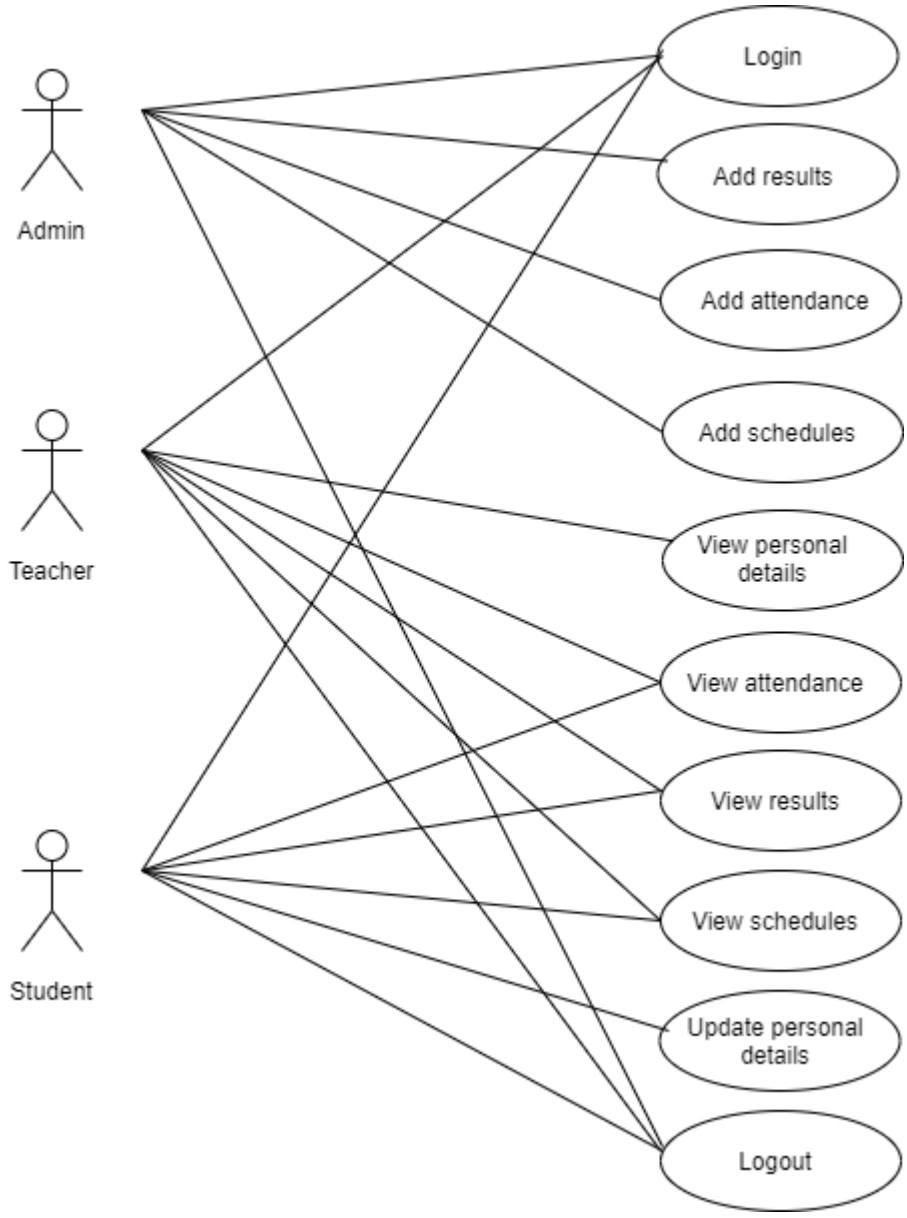
```
Results extracted to build use cases informations

'admin' role has access to the following actions:
    Changepassword
    Changequestion
    DonatorForm
    DeleteDonator
    ViewDonator
    ViewAllDonatorsAction
    AddSoilType
    UpdateProfile
    UpdateProfile
    ViewInbox
    ViewOutbox
    ViewReportPresent
    ViewUsersByDateForm
    Changepassword
    Changequestion
    ViewUsers
    LogoutAction

'ngo' role has access to the following actions:
    LogoutAction
    AddMarketReport
    traing
    traing_view
    NViewQueries
    PostAnswer
    NViewInbox
    NViewOutbox
    Nsearch
    Changepassword
    Changequestion

'user' role has access to the following actions:
    Changepassword
    Changequestion
    qryanswers
    PostAnswer
    traing
    traing_view
    reqtraining
    LogoutAction
    crop_season
    crop_fert
    cmdwiserpt
    cmdmktwisedlystate
    mktwisemonthly
    regngos
    UViewInbox
    UViewOutbox
    Usearch
    AddMarketReport
```

Fonte: Elaborado pela Autora (2021)

Na comparação feita entre as documentações apresentadas nas Figuras 17 e 18, pode-se perceber que nos resultados obtidos há um ator a menos. Além disto, outras ações presentes no diagrama não foram encontradas de forma explícita na lista da Figura 18, como por exemplo *commodity* e *fertilizer*. Novamente, o nível de detalhamento das ações no diagrama de casos de uso, e das ações encontradas nas classes do código-fonte está diferente, é preciso realizar uma interpretação para fazer o relacionamento entre as Figuras 17 e 18.

Como última aplicação de teste de casos de uso neste trabalho, foi utilizado Code With C (2014c), que possui o diagrama conforme a Figura 19. Em seguida, apresenta-se na Figura 20 com os resultados obtidos deste código-fonte.

Figura 19 – Casos de Uso Teste 3



Fonte: Code With C (2014c, p.15)

Figura 20 – Resultado Casos de Uso Teste 3

```
Results extracted to build use cases informations

'admin' role has access to the following actions:
    logout
    AddStudentDetails
    AddFacaultytDetails
    AddTimeTable
    AddNotificationDetails
    UpdateStudentDetails
    UpdateFacultyDetails
    EditTimeTable
    SubjectMerit

'teacher' role has access to the following actions:
    logout
    AddReportCardDetails
    AddStudentRemarks
    Recchange_pass1
    UpdateStudentRemarks
    ViewNotification
    ViewTimeTable


'student' role has access to the following actions:
    logout
    ViewStudentProfile
    ViewNotification
    ViewReport1
    ViewRemarks
    ViewTimeTable
    Recchange_pass
```

Fonte: Elaborado pela Autora (2021)

A comparação entre as Figuras 19 e 20 mostrou que os atores foram extraídos de forma correta, sendo eles: *admin, student* e *teacher*. Em relação às ações obtidas como resultado, estão condizentes com o diagrama original disponibilizado, levando em consideração a diferença de detalhamento das classes do código-fonte.

5.3 CONSIDERAÇÕES DOS RESULTADOS OBTIDOS

Os resultados obtidos, da análise dos códigos-fonte, foram importantes para a validação da eficácia do software desenvolvido. Apesar da falta de representação gráfica dos resultados, foi possível comparar os diagramas providos junto aos código-fonte e as respostas apresentadas de forma textual.

É necessário levar em consideração, que para uma extração eficaz e fiel à documentação original, é preciso que o código-fonte tenha sido escrito de forma clara e com variáveis que sejam condizentes com suas funcionalidades. Caso contrário, as informações obtidas não serão parecidas com as originais. Ainda assim, pode haver diferenças significativas, devido ao nível de detalhe do código para a documentação, ou devido a uma documentação incompleta disponibilizada.

6 CONCLUSÃO

Com este trabalho, foi possível compreender os princípios básicos de uma documentação de software, além de sua importância para um bom andamento de um projeto após sua conclusão. Possuir uma documentação atualizada sobre o software a ser trabalhado, agiliza processos de entendimento e manutenções do mesmo. Ademais, pôde-se aprender mais sobre os conceitos de engenharia reversa e suas possíveis aplicações. Observou-se que diferentes etapas podem ser definidas dentro da engenharia reversa, porém, todas tem o mesmo objeto de extrair informações de algo existente.

Durante o desenvolvimento do software proposto, colocou-se em prática o aprendizado adquirido sobre programação, levantamento de requisitos e análise de código-fonte. Tais conhecimentos foram fundamentais para a construção deste sistema. Como resultado, construiu-se um software capaz de analisar e extrair informações importantes de outros códigos-fonte, atendendo ao objetivo inicial deste trabalho. Provando assim que, com a aplicação dos conceitos de engenharia reversa é possível gerar a documentação de softwares legados.

Encontrou-se dificuldades em definir um padrão para realizar a extração de informações, que atendesse o objetivo proposto neste trabalho. Uma vez encontrado um padrão, fez-se o tratamento do conteúdo dos arquivos em busca de tais padrões. Ainda assim, observou-se que se o código-fonte não for bem elaborado, ou se não possuir nomes de variáveis e classes de forma auto explicativa e fáceis de entender, os resultados podem ser diferentes do esperado. Além disto, não foi possível a criação de elementos gráficos para a representação de diagramas e tabelas, obtendo-se os resultados apenas em formato textual.

Ainda assim, durante os testes de validação, esbarrou-se em casos onde o resultado obtido não condizia exatamente com a documentação original dos códigos-fonte, abrindo margem para uma possível documentação incompleta ou alteração do código-fonte após sua elaboração. Entretanto, apesar de algumas melhorias gráficas serem necessárias, para uma comparação fiel aos documentos originais dos códigos base, pôde-se compreender os resultados obtidos e realizar a comparação de forma visual durante os testes.

Este software desenvolvido pode ser aperfeiçoado com o intuito de ser mais genérico, porém, alguns detalhes podem ficar defasados devido à qualidade do código-fonte a ser analisado, conforme comentado anteriormente. Ou também, pode-se adaptá-lo para sua utilização em apenas um tipo de código-fonte, definindo certas particularidades do sistema a ser analisado. Desta forma, os resultados serão mais precisos, e sua utilização pode ser realizada com certa frequência para manter a documentação de um software em específico atualizada.

Como trabalhos futuros deste projeto, ficam o seu aperfeiçoamento para torna-lo genérico ou sua customização à uma aplicação específica.

Foi visto ainda, que é preciso realizar testes exaustivos, para melhor validação do quão genérico o sistema desenvolvido está. Outra forma de melhoramento deste trabalho, está no detalhamento da análise e construção dos resultados, possibilitando uma abordagem de nível baixo e detalhista das respostas encontradas. Também, é possível abranger um número maior de tipos de respostas geradas, permitindo que outros documentos possam ser gerados, além da criação de elementos gráficos.

Concluiu-se que os resultados obtidos com este trabalho foram satisfatórios, realizando-se a solução proposta e comprovando a possibilidade de sua aplicação.

REFERÊNCIAS

ÁVILA, Lucas Tadeu Farias de. **Uma Proposta de Engenharia Reversa Baseada em Testes de Software.** 2013

CAGNIN, Maria I.; PENTEADO, Rosângela A. D. **Avaliação das vantagens quanto à facilidade de manutenção e expansão de sistemas legados sujeitos à engenharia reversa e segmentação**. São Carlos, 2000. Disponível em: http://www2.sbc.org.br/csbc2000/pdf/ctd/dissertacao/Lugar02Mestrado.pdf/ Acesso em 26 nov. 2020.

CHIKOFSKY, Elliot J.; CROSS II, James H. **Reverse Engineering and Design Recovery:** A Taxonomy. IEEE Software, v.7, n.1, p.13-17, 1990.

CODE WITH C. **Bug Tracking System Java Project.** Publicado em: 6 nov. 2014a. Disponível em: https://www.codewithc.com/bug-tracking-system-java-project/ Acesso em: 4 mai. 2021.

CODE WITH C. **Career Information Management System Java Project.** Publicado em: 7 out. 2014b. Disponível em: https://www.codewithc.com/career-information-management-system-java-project/ Acesso em: 4 mai. 2021.

CODE WITH C. **Farmers Buddy Java Project.** Publicado em: 1 mai. 2015. Disponível em: https://www.codewithc.com/farmers-buddy-java-project/ Acesso em: 5 mai. 2021.

CODE WITH C. **Student Result Processing System Java Project.** Publicado em: 17 out. 2014c. Disponível em: https://www.codewithc.com/student-result-processing-system-java-project/ Acesso em: 5 mai. 2021.

COSTA, Rejane M. **FUSION-RE/I:** Um método de Engenharia Reversa para Auxiliar a Manutenção de Software. Dissertação (Mestrado), ICMC-USP, São Carlos, 1997. 100p

FELTRIM, Valéria Delisandra. **Apoio à Documentação de Engenharia Reversa de Software por meio de Hipertextos.** 1999. Dissertação (Mestrado em Ciências – Área de Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação – Universidade de São Paulo, São Carlos, 1999

KOSCIANSKI, André; SOARES, Michel dos S. **Qualidade de Software:** aprenda as metodologias e técnicas mais modernas para o desenvolvimento de software. 2° ed. São Paulo: NOVATEC, 2007.

MAGALHÃES, Luís Paulo A. **Estudo Sobre Engenharia Reversa e Avaliação da Usabilidade de Ferramentas Case para Engenharia Reversa de Software.** Monografia em Ciência da Computação Universidade Federal de Lavras, Lavras, 2008.

MDN WEB DOCS. **JavaScript.** Publicado em: 4 mai. 2021 Disponível em: https://developer.mozilla.org/en-US/docs/Web/JavaScript/ Acesso em: 15 abr. 2021.

MORESI, Eduardo. **Metodologia de Pesquisa**. Programa de Pós Graduação Stricto Sensu em Gestão do Conhecimento e Tecnologia da Informação da Universidade Católica de Brasília, Brasília, 2003.

MOURA, Ronildo Cesar. **O Uso da Engenharia Reversa no Desenvolvimento de Software Seguro.** Dissertação (Mestrado), ESAB – Escola Superior Aberta do Brasil, Vitória, 2008.

NODE.JS. **About Node.js.** Disponível em: https://nodejs.org/en/about/ Acesso em: 15 abr. 2021.

PENTEADO, Rosângela D.; GERMANO, Fernão S. R.; MASIERO, Paulo C. **An Overall Process Based on Fusion to Reverse Engineer Legacy Code.** IEEE Software, 1996.

PERES, Darley R. et al. **Padrões de Processo para a Engenharia Reversa baseado em Transformações.** SugarloaPLop Conference, 2003.

PRESSMAN, Roger S. **Engenharia de Software:** uma abordagem profissional. 7° ed. Porto Alegre: AMGH, 2011.

SOMMERVILLE, Ian. **Engenharia de Software**. 9° ed. São Paulo: PEARSON, 2011.

SOUSA, Henrique Prado; LEITE, Julio Cesar Sampaio do Prado. **Aplicação de Engenharia Reversa e Reengenharia de Software no Desenvolvimento de plugins para a Ferramenta Oryx.** Monografia em Ciência da Computação PUC, Rio de Janeiro, 2012.

VISUAL STUDIO CODE. **Getting Started.** Publicado em: 2021. Disponível em: https://code.visualstudio.com/docs Acesso em: 15 abr. 2021.