MINISTÉRIO DA EDUCAÇÃO

SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA

INSTITUTO FEDERAL GOIANO – CAMPUS MORRINHOS

CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

THIAGO BATISTA DE JESUS

ENGENHARIA REVERSA DE SOFTWARE LEGADO

Morrinhos-GO

Março– 2023

MINISTÉRIO DA EDUCAÇÃO

SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA

INSTITUTO FEDERAL GOIANO – CAMPUS MORRINHOS

CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET

THIAGO BATISTA DE JESUS

ENGENHARIA REVERSA DE SOFTWARE LEGADO

Trabalho de curso apresentado ao Instituto Federal, Ciência e Tecnologia Goiano – Campus Morrinhos, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas Para Internet, sob orientação do Professor Norton Coelho Guimarães.

Morrinhos-GO

Março– 2023

Sistema desenvolvido pelo ICMC/USP
Dados Internacionais de Catalogação na Publicação (CIP)
Sistema Integrado de Bibliotecas - Instituto Federal Goiano

B58e

BATISTA DE JESUS, THIAGO
ENGENHARIA REVERSA DE SOFTWARE LEGADO / THIAGO BATISTA DE JESUS; orientador Norton Coelho Guimarães. -- Morrinhos, 2023.
33 p.

TCC (Graduação em CURSO DE TECNOLOGIA EM SISTEMAS PARA INTERNET) -- Instituto Federal Goiano, Campus Morrinhos, 2023.

1. ENGENHARIA REVERSA. 2. SOFTWARE LEGADO. I. Coelho Guimarães, Norton , orient. II. Título.

SERVIÇO PÚBLICO FEDERAL
MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA GOIANO

Ata nº 16/2023 - CCEPTNM-MO/CEPTNM-MO/DE-MO/CMPMHOS/IFGOIANO

ATA DE DEFESA DE TRABALHO DE CURSO

Aos 13 dias do mês de abril de 2023, às 19 horas e 15 minutos, reuniu-se a banca examinadora composta pelos docentes: Norton Coelho Guimarães (orientador), José Pereira Alves (membro), Hiury Luiz dos Santos (membro), para examinar o Trabalho de Curso intitulado "Engenharia Reversa de Software Legado" do(a) estudante Thiago Batista de Jesus, Matrícula nº 2016104211710271 do Curso de Tecnologia em Sistemas para Internet do IF Goiano – Campus Morrinhos. A palavra foi concedida ao estudante para a apresentação oral do TC, houve arguição do candidato pelos membros da banca examinadora. Após tal etapa, a banca examinadora decidiu pela **APROVAÇÃO** do estudante. Ao final da sessão pública de defesa foi lavrada a presente ata que segue assinada pelos membros da Banca Examinadora.

(Assinado Eletronicamente)

Norton Coelho Guimarães

Orientador(a)

(Assinado Eletronicamente)

José Pereira Alves

Membro

(Assinado Eletronicamente)

Hiury Luiz dos Santos

Membro

Documento assinado eletronicamente por:

- **Jose Pereira Alves**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 13/04/2023 20:04:36.
- **Hiury Luiz dos Santos**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 13/04/2023 20:02:42.
- **Norton Coelho Guimaraes**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 13/04/2023 20:01:06.

Este documento foi emitido pelo SUAP em 13/04/2023. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse https://suap.ifgoiano.edu.br/autenticar-documento/ e forneça os dados abaixo:

Código Verificador: 486172
Código de Autenticação: b618d75b50

INSTITUTO FEDERAL GOIANO
Campus Morrinhos
Rodovia BR-153, Km 633, Zona Rural, None, None, MORRINHOS / GO, CEP 75650-000
(64) 3413-7900

Agradecimentos

Gostaria de iniciar os agradecimentos, agradecendo a Deus que me deu forças e ânimo de conseguir chegar ao final de mais essa etapa em minha vida.

A minha família que me apoiou e deu ânimo para que não desistisse, continuando assim essa jornada, aos meus professores que com muito zelo nos passaram o conhecimento para que pudesse chegar aqui, aos meus colegas de curso, os quais estiveram ao meu lado dividindo os conhecimentos e dificuldades, completando assim uns aos outros, sendo assim agradeço a todos que de certa forma fizeram parte dessa jornada.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 - INTRODUÇÃO

Com a avanço cada vez mais rápida da tecnologia, conseguimos ver que dependemos de soluções que consigam acompanhar essa evolução, com essa necessidade cada vez sendo mais ágil, nos deparamos com *softwares* que por não serem desenvolvidos com as atuais tecnologias acabam não suportando manutenções, sendo assim não conseguem acompanhar a demanda do mercado.

Por mais que para o período que foi desenvolvido ele atendesse as expectativas, com o passar do tempo e as constantes mudanças de procedimentos dentro de algumas corporações, ou até mesmo as leis do nosso país que sofrem alterações, por diversas vezes esses sistemas legados não têm tecnologias suficientes para que possam acompanhar essa evolução.

Ao insistir em realizar modificações em um *software* defasado, podemos ter vários prejuízos sistêmicos como: não conseguir manter uma lógica, ser mal projetado, correção de erros existentes, a tecnologia não acompanhar, e a falta de documentação acaba dificultando esse processo.

Mesmo que esse sistema legado esteja defasado, não conseguindo acompanhar a necessidade de seus usuários, em sua grande maioria são fundamentais e de grande utilização, sendo assim a necessidade de se passar por uma reconstrução do absoluto zero, mais acompanhando as suas premissas.

Tendo como exemplo o sistema existente, conseguimos analisar suas funcionalidades, verificando as possíveis novas implementações, pontos de melhoria, e consequentemente verificando quais tecnologias utilizar para esse novo projeto, esse processo se denomina de Reengenharia de Software Legado.

Nesse Trabalho de conclusão de curso tem como objetivo realizar um projeto de Reengenharia de Software Legado de uma aplicação desenvolvida no Access 2010, denominado “SGM” Sistema Gestão Manutenção, que foi desenvolvida para uma empresa que possuía frota própria, que tinha a necessidade de ter o controle de suas manutenções e históricos das mesmas realizadas em um grande período de tempo.

Com o passar do tempo essa aplicação ficou obsoleta, não conseguindo acompanhar a evolução da tecnologia, e o volume de dados que ela armazenava já não era o suficiente, com essas informações podemos ver que precisaria passar por um processo de reengenharia, a fim de deixar o sistema mais ágil, com uma maior capacidade de armazenamento, e melhorar a sua usabilidade, deixando de ser um

sistema instalado em um computador de mesa, podendo ser utilizado de qualquer máquina que possua acesso à internet e um navegador, se tornando um sistema web.

Neste Trabalho de Conclusão de Curso será abordado nos primeiros capítulos conceitos de processos de reengenharia reversa e alguns exemplos apresentados por pesquisadores, como modelos de processos entre outros. Em sequência veremos o estudo de caso, tendo como base a Reengenharia Reversa do SGM.

2 - ENGENHARIA DE *SOFTWARE* - EVOLUÇÃO

Hoje em dia a maioria das empresas estão migrando seus sistemas para novas linguagens, para trazer uma melhor qualidade e agilidade para suas aplicações. Essa prática está se tornando mais comum devido ao volume de dados arquivados e os problemas em se dar manutenção nos *softwares* mais antigos, por não acompanharem as evoluções da tecnologia (Sommerville, 2011).

A Engenharia de *Software* vem como uma área da computação, focada no desenvolvimento e manutenção de softwares, ela que mostra o caminho a ser trilhado para os desenvolvedores e engenheiros de *softwares*, seguindo as necessidades apresentadas pelos clientes. Em conclusão, a Engenharia de *Software* é a base para o início de criação de um sistema independentemente de qual for (Sommerville, 2011).

Segundo (Sommerville, 2011) “consequentemente, a maioria das grandes empresas gasta mais na manutenção de sistemas existentes do que no desenvolvimento de novos sistemas. Baseado em uma pesquisa informal da indústria, Erlikh (2000) sugere que 85% a 90% dos custos organizacionais de *software* são custos de evolução.”

Em sua grande maioria *software* ou aplicações que se tem utilidade tem um tempo de uso muito longo. Contando-se que o valor para o desenvolvimento de um *software* é muito alto, a maioria das empresas que tem um sistema desenvolvido exclusivamente para ela, tem a necessidade de recuperar o dinheiro investido no mesmo, por esse motivo a evolução e manutenção é necessária. Quando pensamos na engenharia de um *software* entendemos que é um processo que tem a duração de toda a vida útil do sistema (Sommerville, 2011).

De acordo com Sommerville (2011), a dinâmica e evolução de programas é o que estuda a mudança de sistema. Na década de 1970 e 1980, Lehman e Belady (1985) realizaram vários estudos empíricos sobre a mudança de sistema com intenção de compreender mais sobre as características de evolução de software.

Como citado acima, referente a mudanças e evoluções das leis que se obriga a se aprimorar e evoluir, o sistema tem como base as Leis de Lehman (Lehman e Belady, 1985).

Vale lembrar que essas leis são válidas para *softwares* que são desenvolvidos para receberem manutenções, podemos entender o processo e melhor na (Tabela 1.0 Leis de Lehman).

Lei	Descrição
Mudança contínua	Um programa usado em um ambiente do mundo real deve necessariamente mudar, ou se torna progressivamente menos útil nesse ambiente.
Aumento da complexidade	Como um programa em evolução muda, sua estrutura tende a tornar-se mais complexa. Recursos extras devem ser dedicados a preservar e simplicar a estrutura.
Evolução de programa de grande porte	A evolução de programa é um processo de autorregulação. Atributos de sistema como tamanho, tempo entre *releases* e número de erros relatados são aproximadamente invariáveis para cada *release* do sistema.
Estabilidade organizacional	Ao longo da vida de um programa, sua taxa de desenvolvimento é aproximadamente constante e independente dos recursos destinados ao desenvolvimento do sistema.
Conservação da familiaridade	Durante a vigência de um sistema, a mudança incremental em cada *release* é aproximadamente constante.
Crescimento contínuo	A funcionalidade oferecida pelos sistemas tem de aumentar continuamente para manter a satisfação do usuário.
Declínio de qualidade	A qualidade dos sistemas cairá, a menos que eles sejam modificados para refletir mudanças em seu ambiente operacional.
Sistema de *feedback*	Os processos de evolução incorporam sistemas de *feedback* multiagentes, *multiloop*, e você deve tratá-los como sistemas de *feedback* para alcançar significativa melhoria do produto.

Tabela 1: Leis de Lehman (1985)

3 - MANUTENÇÃO DE *SOFTWARE*

Dentro da manutenção de *software* temos como processo geral de evolução e mudança de um sistema que foi liberado para uso. Ao estar em uso um *software* pode apresentar alguns erros ou algumas funcionalidades que não eram esperadas pelo cliente, nesse caso é comunicado a empresa que foi responsável pelo desenvolvimento e Engenharia do Software, para que execute as alterações desejadas (Sommerville, 2011).

Segundo Pressman, Maxim (2016 p. 736) outra razão para problema de manutenção de software é a modalidade dos profissionais. É provável que a equipe (ou pessoa) responsável pelo trabalho original não esteja mais por perto. Pior ainda, outras gerações de profissionais de software podem ter modificado o sistema e já se foram. E hoje pode não ter restado ninguém que tenha conhecimento direto do sistema legado.

A manutenção e mudanças são inevitáveis quando vemos o crescimento da tecnologia e da necessidade e exigência dos clientes, portanto podemos ver a necessidade de desenvolver mecanismos para podermos avaliar e controlar as modificações que serão necessárias no *software* (Sommerville, 2011).

Dentro dessas modificações temos três tipos de manutenção de *softwares* mais comuns, sendo a correção de defeitos a primeira, com acompanhamento dos erros de codificação que tem um custo menor para efetuar as correções, já erros de projeção mais caros, porque por esse motivo implica-se a reescrever várias partes do sistema, se forem erros de requisitos também fica na linha do custo mais alto, pelo motivo que pode ser necessário ter que projetar o sistema ou parte dele novamente. O segundo fica a cargo da adaptação ambiental, que por sua vez passa a ser indispensável a mudança de alguns aspectos do ambiente do sistema, estando entre eles o hardware, a plataforma operacional ou a necessidade de um *software* de apoio sofrer uma mudança. Na terceira posição vem a adição de funcionalidades, essa manutenção já vem quando houve mudança nos requisitos do sistema (Lehman e Belady, 1985).

Vemos que mudanças e manutenções são inevitáveis e temos que acompanhar de forma a termos um custo mais baixo e agilidade para alcançar as soluções, pensando na satisfação do cliente e na produtividade, podendo ser mudanças organizacionais ou de negócio (Sommerville, 2016).

Sendo mais bem visualizado na figura 1.

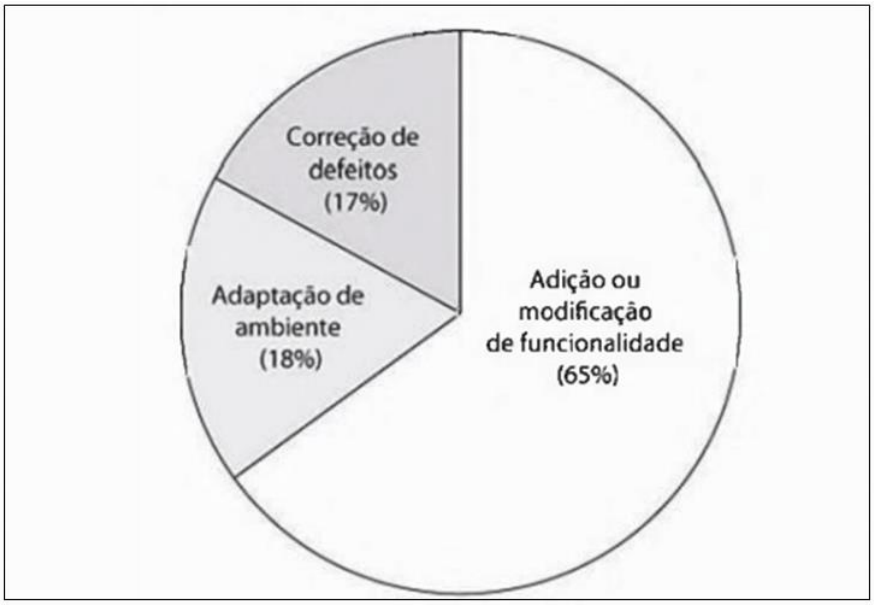

Figura 1: Distribuição do esforço de manutenção (Presman e Maxin, 2016).

4 - REENGENHARIA DE *SOFTWARE* E ENGENHARIA REVERSA

A Reengenharia de *Software* se faz necessária quando nos deparamos com uma aplicação ou sistema que se torna ultrapassado ou com dificuldades de se executar manutenções, mas mesmo assim o *software* continua sendo usado e de grande utilidade para a organização, então, nos deparamos com a necessidade de refazer o sistema. Esse processo de reengenharia vem da aplicação existente, essas informações podem vir do código fonte da interface ou até de aplicações desenvolvidas no *Access* ou até mesmo de planilhas eletrônicas com códigos VBA que ainda são muito utilizadas em muitas empresas, tendo como base essas informações são abstraídas, podendo assim iniciar a reengenharia do software (Lehman e Belady, 1985).

Se formos analisar pelo nosso pouco conhecimento de sistemas em empresas que trabalhamos, e vendo a grande demanda do mercado de tecnologia com inovações a todo tempo, conseguimos identificar facilmente a necessidade de reengenharia por que a aplicação deve continuar em processo de evolução, mais dependendo da linguagem que foi desenvolvida não suporta mais informações ou inserções de novos requisitos, com por exemplo linguagens de segunda geração (Sommerville, 2016).

Segundo Pressman, Maxim (2016 p. 802), o s*oftware* que não pode ser mantido não é um problema novo. Na verdade, a ênfase cada vez maior na reengenharia de software foi gerada pelos problemas de manutenção de quase meio século.

Podemos ter uma noção maior como nos mostra na figura 2.0.

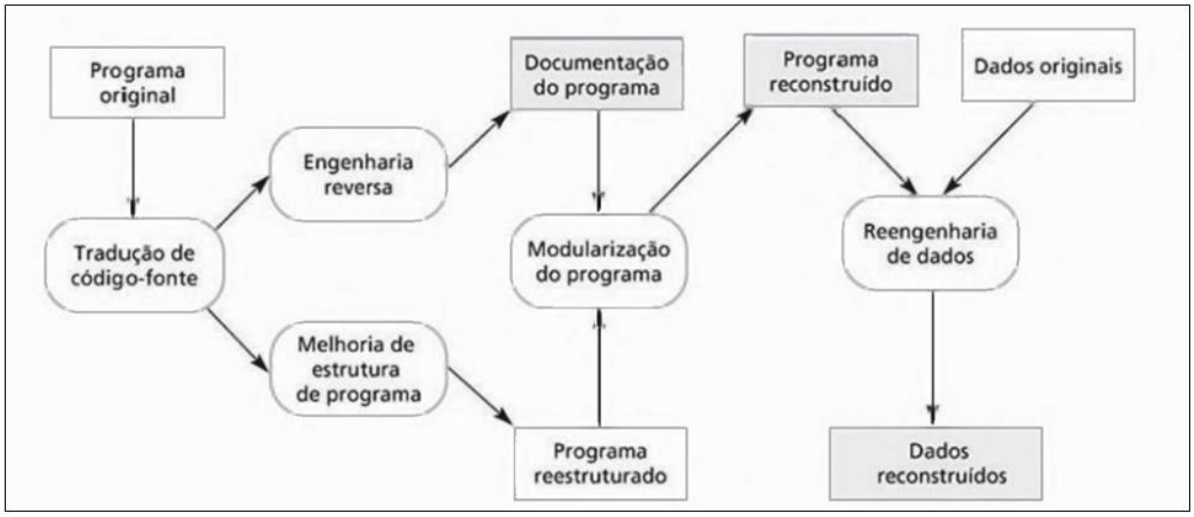

Figura 2: Processo de reengenharia (Presman, Maxin 2016).

Como havíamos falado anteriormente sobre o processo de evolução de um *software* envolve entendermos quando um sistema não pode mais sofrer alterações e nem manutenções. Mesmo assim conhecendo sistemas legados, no caso os mais

velhos são de difícil compreensão para poderem ser mudados, por conta de com passar do tempo foi sofrendo tantas manutenções que às vezes falta clareza para entender o que foi feito (Sommerville, 2016).

Para conseguir manter esses sistemas legados com mais facilidade se faz necessário a reengenharia, que vem envolvendo desde o princípio da documentação, mudança de linguagem, arquitetura entre outros. Em sua grande maioria a funcionalidade do *software* não é alterada (Sommerville, 2016).

4.1 - Benefício da Reengenharia do Software

Podemos citar dois benefícios da reengenharia, o primeiro ao invés da substituição, reduzir o risco, ao desenvolver um *software* do zero existe uma grande possibilidade em fazer novamente um *software* crítico de negócio. Podendo ocorrer erros na especificação ou ter problemas no desenvolvimento. Ocasionando atrasos no início do novo *software* podendo ter custos adicionais e possível perda do negócio. O segundo benefício é a redução de custos pode ser significativamente menor do que o desenvolvimento de um *software* novo (Sommerville, 2016)..

Como exemplo em Sommerville, 2016 e Ulrich (1990) temos um sistema comercial cujo custos de re-implantação cujo custos de implantação foram estimados em 50 milhões de dólares. O sistema foi reconstruído com sucesso por 12 milhões de dólares. Suspeitos que a tecnologia moderna de *software*, o custo relativo de re-implementar é provavelmente inferior a esse, mais ainda consideravelmente superior aos custos da reengenharia.

4.2 - Modelo de Processo

Como sequência se tem visto o modelo de processo de reengenharia de software, Figura 3, se tem por base que a reengenharia toma tempo, com um custo financeiro alto e absorve recursos que poderiam ser usados em outras necessidades mais imediatas (Pressman e Maxim 2016).

Para Pressman, Maxim (2016) existem alguns princípios básicos que seguem o modelo de processo:

- Criar uma lista de critérios para que a inspeção fosse sistemática.
- Remodelar, traz um custo mais baixo em menos tempo.
- Não descartar informações pois elas têm utilidade quando se inicia a reconstrução do *software*.
- Usar tecnologia estando um passo à frente com as inovações do mercado para evitar manutenções a longo prazo.

- Usar práticas que resultem na mais alta qualidade.

Podemos ver o modelo de processo de reengenharia de software que define uma sequência de atividades, para podermos seguir esses princípios e podermos implementá-los. Sendo que essas atividades podem acontecer em sequência, mas nem sempre isso acontece.

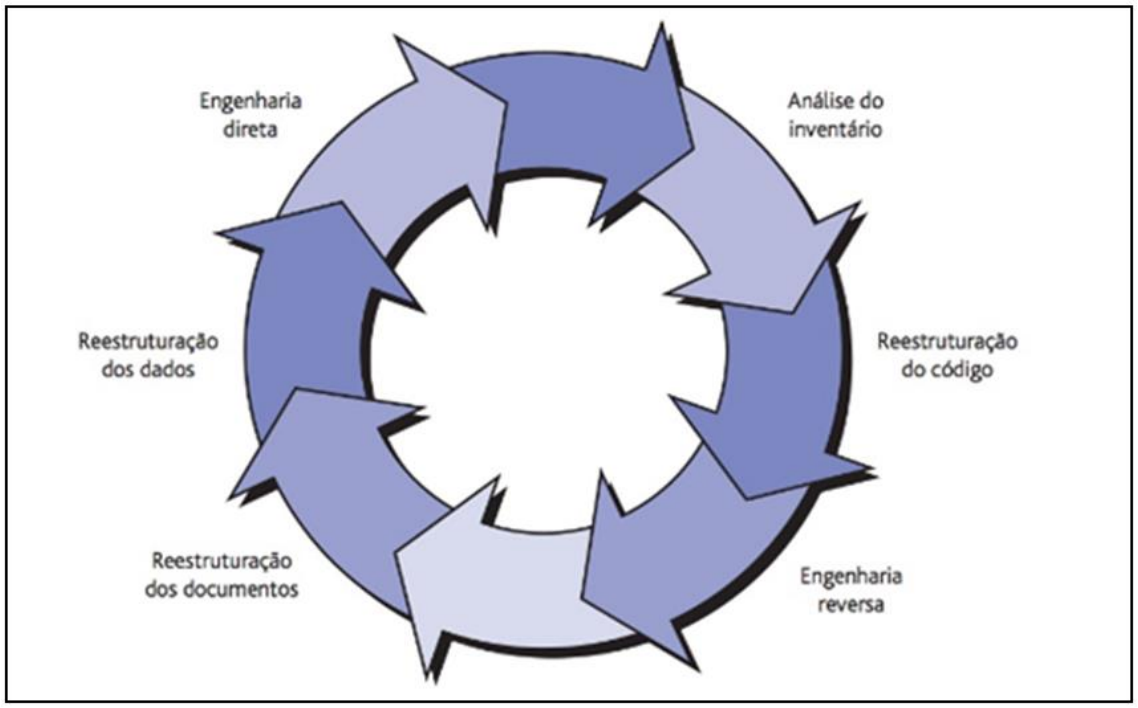

Figura 3: Um modelo de processo de reengenharia de *software* (Presman, Maxin 2016).

Explicando um pouco do que acontece na Figura 3, segue um exemplo de modelo cíclico, ou seja qualquer uma das atividades na sequência que podem ser visitadas e revisitadas. Sendo em qualquer ciclo, pode se terminar o processo. (Pressman e Maxim, 2016):

- Análise de inventário. Deve se ter um inventário de todas as aplicações. Pode ser até mesmo em uma planilha simples, sendo com informações detalhadas, para cada aplicação ativa. Mantendo as informações ordenadas essas informações de acordo com a criticidade do negócio. O mesmo deve ter um acompanhamento regularmente.
- Reestruturação do documento. No caso a documentação fica a critério do sistema a ser documentado, pois para um software que não tenha nenhum tipo de documentação pode ser muito caro, então na sequência tudo que for feito de alterações deve-se documentar, na sua grande maioria é fundamental que o sistema seja todo documentado, atingindo o mínimo essencial.

- Engenharia Reversa. É seguir com base em um programa existente, para se recuperar o projeto analisando o sistema, criando uma abstração maior que a do código-fonte, ou de uma aplicação que possua o mínimo de código.
- Reestruturação do código. No caso de reestruturação do código em sistemas legados, costumam ser bem sólidos, mas as codificações dificultam o entendimento e a manutenção, por estarem em modelos defasados.
- Reestruturação dos dados. Em sua grande maioria a reestruturação dos dados começa na engenharia reversa, essa reestruturação é a atividade de engenharia completa.
- Engenharia direta. A engenharia direta recupera informações do *software* existente, como também utiliza-se dessas informações para alterá-lo e aumentar o desempenho e acrescentar novas funções.

Em um processo de desenvolvimento de *software*, os estágios iniciais envolvem conceitos mais gerais, independentes da implementação. O aumento de detalhes durante o processo de desenvolvimento conceitua os níveis de abstração. Estágios iniciais do sistema planejam e definem requisitos de alto nível quando comparados com a própria implementação. Essa comparação é importante para deixar claro que nível de abstração e grau de abstração são grandezas distintas. Enquanto o nível de abstração é um conceito que diferencia os estágios conceituais do projeto, o grau de abstração é intrínseco a cada estágio (Pressman e Maxim 2016).

4.3 - Engenharia Reversa

Sommerville (2011), menciona que dentro da engenharia reversa o sistema é analisado e as informações são extraídas a partir dele. Isso ajuda a documentar sua organização e funcionalidade. Esse processo também é completamente automatizado.

Pressman e Maxim (2016), repassa que a engenharia reversa pode extrair informações do projeto a partir do código-fonte, mas o nível de abstração, a completude da documentação, o grau segundo o qual as ferramentas e um analista humano trabalham juntos e a direcionalidade do processo são altamente variáveis.

Dando continuidade Pressman, Maxim(2016) afirma que o nível de abstração de um processo de engenharia reversa e as ferramentas utilizadas para realizá-lo refere-se à sofisticação das informações de projeto que podem ser extraídas do código fonte. Idealmente o nível de abstração deve ser o mais alto possível.

Em muitos casos a dificuldade da engenharia reversa diminui conforme o nível de abstração aumenta. Dependendo da quantidade de código-fonte, é um pouco mais fácil desenvolver uma representação completa do projeto. Também podem ser resultado

de representações simples do projeto arquitetural. Mas segundo Pressman, Maxim(2016) é mais difícil desenvolver um conjunto completo de diagramas UML ou modelos. Na Figura 4 explica melhor o processo de engenharia reversa.

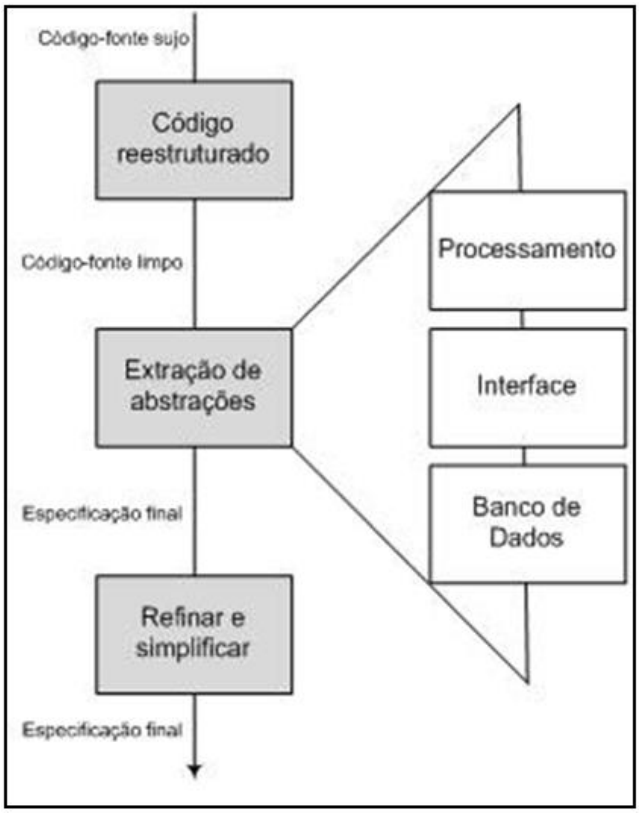

Figura 4: Processo de Engenharia Reversa (Presman e Maxin, 2016).

Conseguimos entender conforme ilustra figura acima que o processo de engenharia reversa começa com o código fonte sujo que como exemplo seria o nosso sistema legado que iremos trabalhar, aplicando nele a engenharia reversa, em sequência o código reestruturado e em seguida o código fonte limpo que já é retirado as redundâncias o que não tem necessidade e o excesso de código, em continuidade a extração de abstrações dentro dela temos que abstrair a parte de processamento, interface e base de dados, depois temos a especificação aonde iremos e refinar e simplificar, esse sistema legado, e por último nessa figura a especificação final do código fonte, esse é um entendimento geral para melhor entender como funciona esse processo (Presman e Maxin, 2016).

Segundo Pressman e Maxim (2016) o centro da engenharia reversa é uma atividade chamada extrair abstrações. Você deve avaliar o programa antigo e, com base no código-fonte (muitas vezes não documentado), desenvolver uma especificação significativa do processamento executado, da interface de usuário aplicada e das estruturas de dados de programas ou do banco de dados utilizadas.

4.4 - Engenharia Reversa de Dados

A engenharia reversa de dados é uma das primeiras atividades a ser executada no processo de reengenharia de *software*. Como parte do processo em níveis de sistemas muitas vezes devem passar por reengenharia para avaliar as estruturas internas. Os arquivos os bacos de dados devem se passar por engenharia reversa para entender o que está defasado do sistema legado e fazer as alterações necessárias para um sistema com comunicação mais rápida, seguindo a acomodação de novos paradigmas, definindo assim a estrutura de dado para o sistema reformulado. As estruturas internas focam na definição de classes em objetos (Pressman e Maxim, 2016).

Segundo Pressman, Maxim(2016) isso é feito examinando-se para o código do programa para agrupar variáveis de programa relacionadas. Em muitos casos, a organização dos dados dentro do código identifica tipos de dados abstratos. Por exemplo, estruturas de registros, arquivos, listas e outras estruturas de dados muitas vezes fornecem indicador inicial das classes.

Pressman, Maxim(2016), afirma que seja qual for a sua organização lógica e estrutura física, um banco de dados permite a definição de objetos de dados e suporta alguns métodos para estabelecer relações entre os objetos. Portanto, para fazer a reengenharia em um esquema de banco dados para outro é necessário entender os objetos e suas relações.

5 - VISÃO GERAL BANCO DE DADOS

O banco de dados nada mais é do que uma junção de dados relacionados. Tendo como exemplos, nomes, números de telefone, documentos, informações e históricos de movimentações entre outros, essas informações são uma junção de dados com um significado implícito, estando anotados em uma agenda ou em planilhas eletrônicas, não deixam de ser um banco de dados (SILBERSCHATZ, 1999).

Para Machado, Abreu (2004), o banco de dados é uma coleção de fatos registrados que refletem o estado de certos aspectos de interesse do mundo real. A todo o momento o conteúdo do banco de dados representa uma visão instantânea do estado do mundo real. Cada mudança em algum item do banco de dados reflete uma mudança ocorrida na realidade.

Segundo Elmasri e Navathe (2005), um sistema gerenciador de banco de dados (SGBD) é uma coleção de programas que permite aos usuários criar e manter um banco de dados. O SGBD é um sistema de *software* que facilita os processos de definição, construção, e manipulação e compartilhamento de banco de dados entre vários usuários e aplicações, conforme Figura 5..

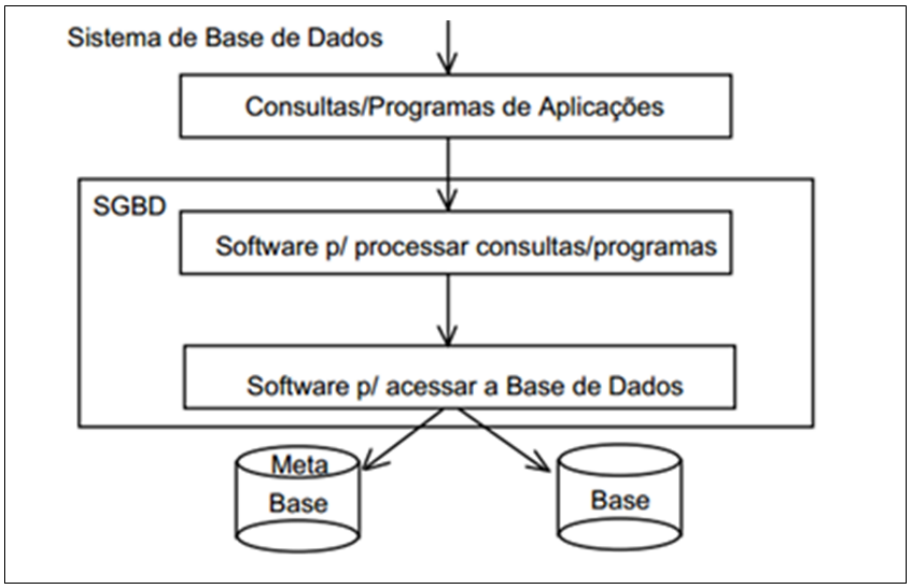

Figura 5: Sistema Base de Dados (Elmasri e Navathe, 2005)

5.1 - Modelo Conceitual e Modelo Entidade e Relacionamento (MER)

Para Machado e Abreu (2004), ao se utilizar Modelagem Conceitual de Dados com a técnica de Entidade e Relacionamento, obteremos resultados e esquemas puramente conceituais sobre as essências de um sistema, ou melhor sobre o negócio para qual estamos desenvolvendo um projeto, não se representando procedimentos ou fluxo de dados existentes.

Segundo Date (2000), o MER é um modelo de dados conceitual de alto-nível, ou seja, seus conceitos foram projetados para serem compreensíveis a usuários, descartando detalhes de como os dados são armazenados. Atualmente, o MER é usado principalmente durante o processo de projeto da base de dados. Existem expectativas para que uma classe de SGBD's baseados diretamente no MER esteja disponível no futuro.

Em geral podemos entender que o Modelo de Entidade e Relacionamento representa uma estrutura que um banco de dados possuirá, claro que o banco de dados poderá conter outras entidades, como tabelas e chaves, que fazem mais sentido no contexto de base de dados relacional. Na Figura 6, podemos ver o esquema geral de modelagem de dados usando MER.

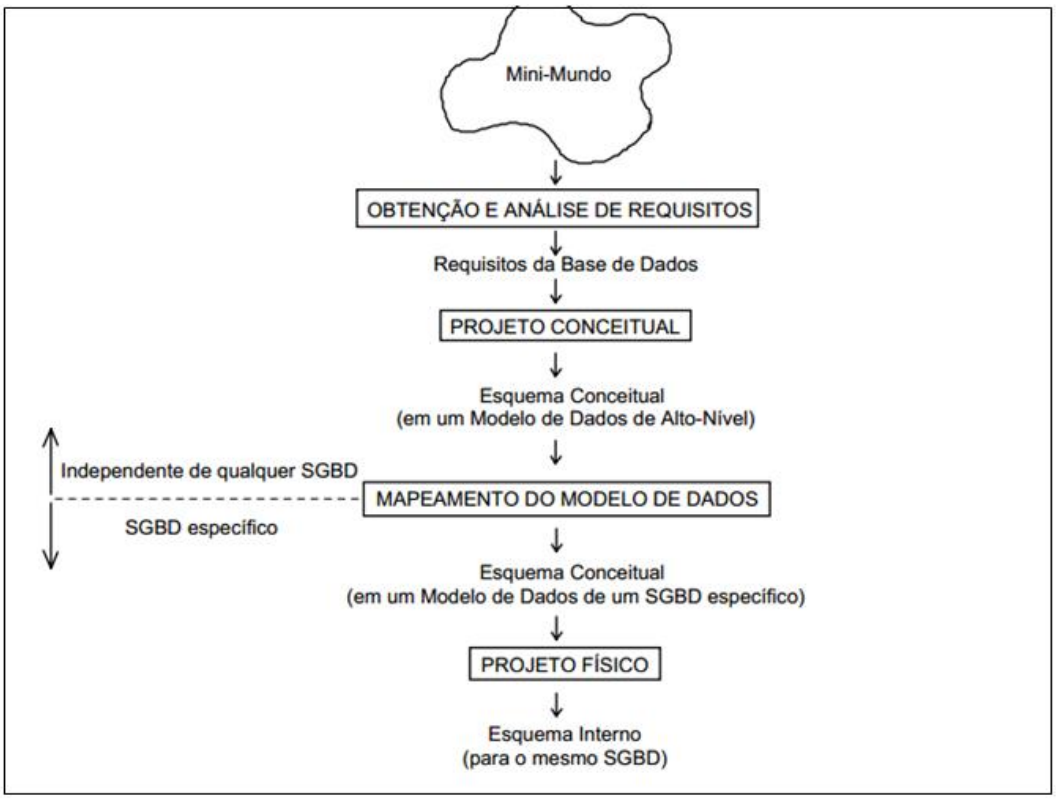

Figura 6: Modelagem de Dados (Date 2000).

Segundo Machado e Abreu (2004), o objetivo da Modelagem de Dados é transmitir e apresentar uma representação única, não redundante e resumida, dos dados de uma aplicação. Em projetos conceituais de aplicações de banco de dados o modelo Entidade Relacionamento é o mais largamente utilizado para a representação e entendimento dos dados que compõem a essência de um sistema.

5.2 - Normalização em Banco de Dados Relacionais

Segundo Dante (2000), o processo de normalização foi proposto por Codd em 1972. Esta é uma maneira mais formal de garantir as diretrizes. Sendo descrito com um processo que se aplicam regras a todas as tabelas do banco de dados, tendo como objetivo evitar erros e falhas no projeto.

Para Machado e Abreu (2004), através deste processo pode-se, gradativamente, substituir um conjunto de entidades e relacionamentos por um outro, o qual se apresenta "purificado" em relação às anomalias de atualização (inclusão, alteração e exclusão) as quais podem causar certos problemas, tais como: grupos repetitivos (atributos multivalorados) de dados, dependências parciais em relação a uma chave concatenada, redundância de dados desnecessária, perdas acidentais de informação, dificuldade na representação de fatos da realidade observada e dependências transitivas entre atributos.

5.2.1 - Primeira Forma Normal (1FN)

Segundo Machado, Abreu(2004), a 1FN diz que: cada ocorrência da chave primária deve corresponder a uma e somente uma informação de cada atributo, ou seja, a entidade não deve conter grupos repetitivos (multivalorados).

Para Dante(2000), a primeira forma normal é agora genericamente considerada como parte da definição formal de uma relação; historicamente foi definida para não permitir atributos multivalorados, compostos e suas combinações.

5.2.2 - Segunda Forma Normal (2FN)

Podemos afirmar que para estarmos na 2FN é preciso estar na 1FN. Como veremos na sequência, todos atributos não chaves da tabela devem depender unicamente da chave primária, e não dependendo apenas de parte dela.

Para Machado e Abreu(2004), a finalidade de tornar ainda mais estável o modelo de dados, a aplicação da 2FN sobre as entidades em observação geram novas entidades, que herdarão a chave parcial e todos os atributos que dependem desta chave parcial, ou seja, uma entidade para estar na 2FN não pode ter atributos com dependência parcial em relação à chave primária.

5.2.3 - Terceira Forma Normal (3FN)

Assim como na 2FN, para estarmos na 3FN precisamos estar na segunda, a 3FN e a junção das três.

Para Machado e Abreu (2004), uma entidade está na 3FN se nenhum de seus atributos possui dependência transitiva em relação a outro atributo da entidade que não participe da chave primária, ou seja, não exista nenhum atributo intermediário entre a chave primária e o próprio atributo observado.Ao retirarmos a dependência transitiva, devemos criar uma nova entidade que contenha os atributos que dependem transitivamente de outro e a sua chave primária é o atributo que causou esta dependência.

6 - VISÃO GERAL DA LINGUAGEM DE MODELAGEM UNIFICADA (UML)

A UML é a sigla de *Unified Modeling Language*, que em português significa linguagem de modelagem unificada, sendo assim ela é uma linguagem. Ou seja, uma linguagem é simplesmente uma ferramenta para expressar uma ideia (Bezerra, 2015).

Para Schach (2010), na qualidade de linguagem, a UML pode ser usada para descrever *software* desenvolvidos por meio de paradigma tradicional ou de qualquer uma das muitas versões do paradigma de orientação a objetos, inclusive o Processo Unificado. Em outras palavras, a UML é uma notação, não uma metodologia. Ela é uma notação que pode ser usada em conjunto com qualquer metodologia.

Dentre da UML existem vários modelos de diagramas, iremos contemplar na sequência somente o diagrama de classes, por ser um dos mais usado (Bezerra, 2015).

6.1 - Modelo Diagrama de Classes

Podemos dizer que do desenvolvimento de um *software*, o diagrama de classes é uma representação das classes que servem de modelo para objetos. A importância de encontrar as classes para o desenvolvimento é por que cada classe vem representando cada tabela do banco de dados (Bezerra, 2015).

Segundo Pádua (2010), na UML, a classe é representada por um retângulo dividido em compartimentos (Figura 7). Os três compartimentos mais usados são aqueles que contêm, respectivamente, o nome da classe, os atributos e as operações. Para maior clareza nos diagramas, pode-se suprir cada um dos compartimentos de atributos e operações, ou deixar de mostrar determinados atributos ou operações. São opcionais também: a indicação da visibilidade por caracteres ou ícones; a assinatura (lista de argumentos e tipo de retorno) das operações; e o tipo, a multiplicidade (número de ocorrências) e o valor dos atributos.

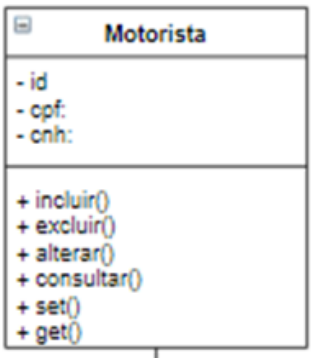

Figura 7: Exemplo de uma Classe Motorista.

7 - ESTUDO DE CASO: ENGENHARIA REVERSA NO SISTEMA GESTÃO DE MANUTENÇÃO

7.1 - A aplicação

Em meados de 2016, foi desenvolvido pelos estudantes do curso Tecnólogo de Sistemas Para Internet, na cidade de Morrinhos-GO, uma aplicação para controle e histórico de manutenções de uma Revenda e Distribuidora de Bebidas com frota própria, operando em dezenove cidades. O objetivo do sistema era controlar as manutenções realizadas nos veículos da frota, conseguindo assim gerar controle e banco de dados, para que com esses dados seja possível realizar análises, e ter um histórico das manutenções realizadas em seus veículos, através de cadastro de motoristas, veículos, equipamentos, suas respectivas manutenções e despesas, a aplicação gerava relatórios de manutenção por veículos, manutenção por motoristas, serviço, controlando horário e data de entrada e saída da oficina, controlando assim a produtividade de execução do serviço.

A aplicação foi desenvolvida com os recursos disponíveis no Microsoft Access (versão 2010), possuindo apenas ligações simples do banco de dados da própria ferramenta, o restante foi feito utilizando as funcionalidades da aplicação para design entre outros. A aplicação funcionava e gerava os resultados esperados para a época. Sendo simples e não possuía documentação, como era de tecnologia ultrapassada não suportaria muitos dados por muito tempo.

7.2 - Necessidade da Reengenharia da Aplicação

Foi necessário a reengenharia da aplicação feita no Microsoft Access devido ser simples e com passar do tempo ficando obsoleta, com capacidade de armazenamento baixa sendo no máximo de 2GB no banco de dados, usando o próprio banco de dados do Access não sendo 100% seguro, tornando a aplicação vulnerável a integridade dos dados, e se levarmos em consideração que a aplicação é de instalação em desktop, gerando um trabalho de comunicação entre outras máquinas, não tendo possibilidade de manutenções que acompanhassem a necessidade, se teve por necessidade a realização da reengenharia, para que fosse possível o acesso remoto em qualquer máquina que possua internet, pelo seu navegador, com uma liberdade e segurança dos dados informados, garantindo uma amarração do banco de dados, e uma padronização clara, para possíveis mudanças que se forem necessárias com a evolução dos processos internos, tendo possibilidade de manutenção no sistema.

7.3 - Reengenharia do Sistema Gestão de Manutenção

Pegando como exemplo o que cita acima Pressman e Maxim (2016), foi adotado o modelo de reengenharia na descritos como análise de inventário, reestruturação de documentos, engenharia reversa, e reestruturação de dados. Não foi seguida essa sequência descrita, pois o modelo como exemplificado na Figura 3, o modelo de processo de reengenharia de software, exemplifica que cada uma dessas atividades pode ser revisada em um ciclo.

Acompanhando esse ciclo, e conhecendo a aplicação a ser trabalhada com a engenharia reversa poderemos ver na Figura 8 a representação da interface antiga do sistema a ser trabalhado, na análise de inventário não houve mudanças significativas, pelo motivo que a sua essência não sofreu nenhuma alteração.

Figura 8: Tela antiga Sistema Gestão de Manutenção

Vamos observar (Tabela 3) quais tecnologias foram utilizadas para realização desse projeto, trazendo assim uma facilidade para entender o processo de reengenharia de *software* e conseguir chegar ao resultado e análise concluindo assim essa reengenharia.

TECNOLOGIAS UTILIZDAS	
Modelagem UML	Draw.io
Modelagem Banco de Dados	Draw.io
Configuração Texto	Word 2010
Desenhar modelo nova interface	Corel Draw

Tabela 3: Tecnologias Utilizadas

Após os estudos tivemos as propostas das notas telas, como podemos visualizar nas Figuras 9, os menus de Cadastros e na Figura 10, os menus de relatórios

Posteriormente, com base no modelo existente no Access (Figura 11) tivemos a aplicação dos conceitos descritos para elaboração do MER, como podemos visualizar na Figura 12, o resultado.

Figura 9: nova tela Sistema Gestão de Manutenção Cadastro

Figura 10: nova tela Sistema Gestão de Manutenção Relatórios

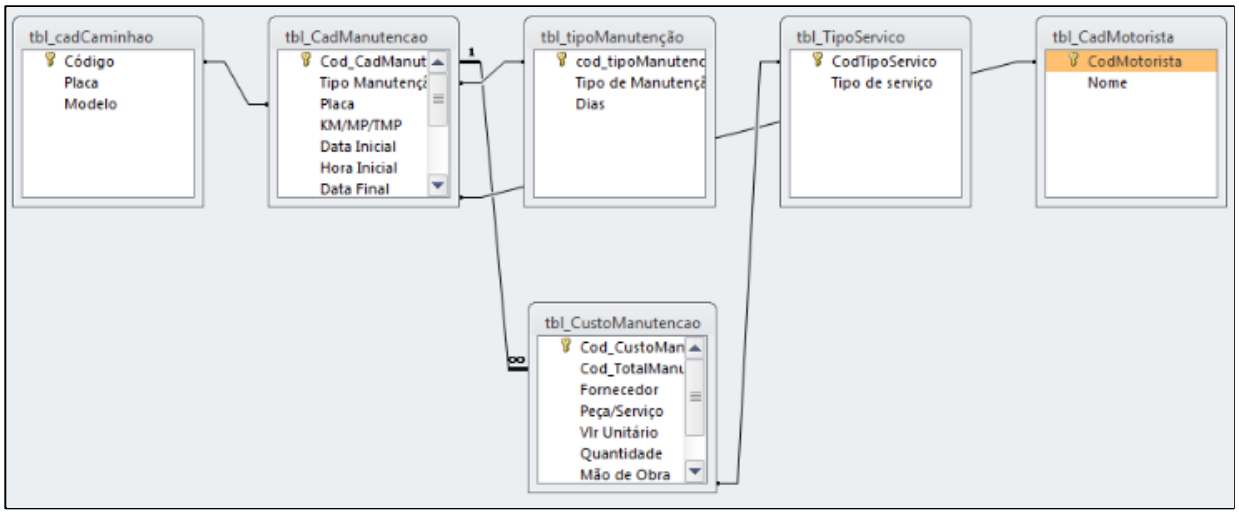

Figura 11: Base de dados Access no Sistema de Gestão de Manutenção.

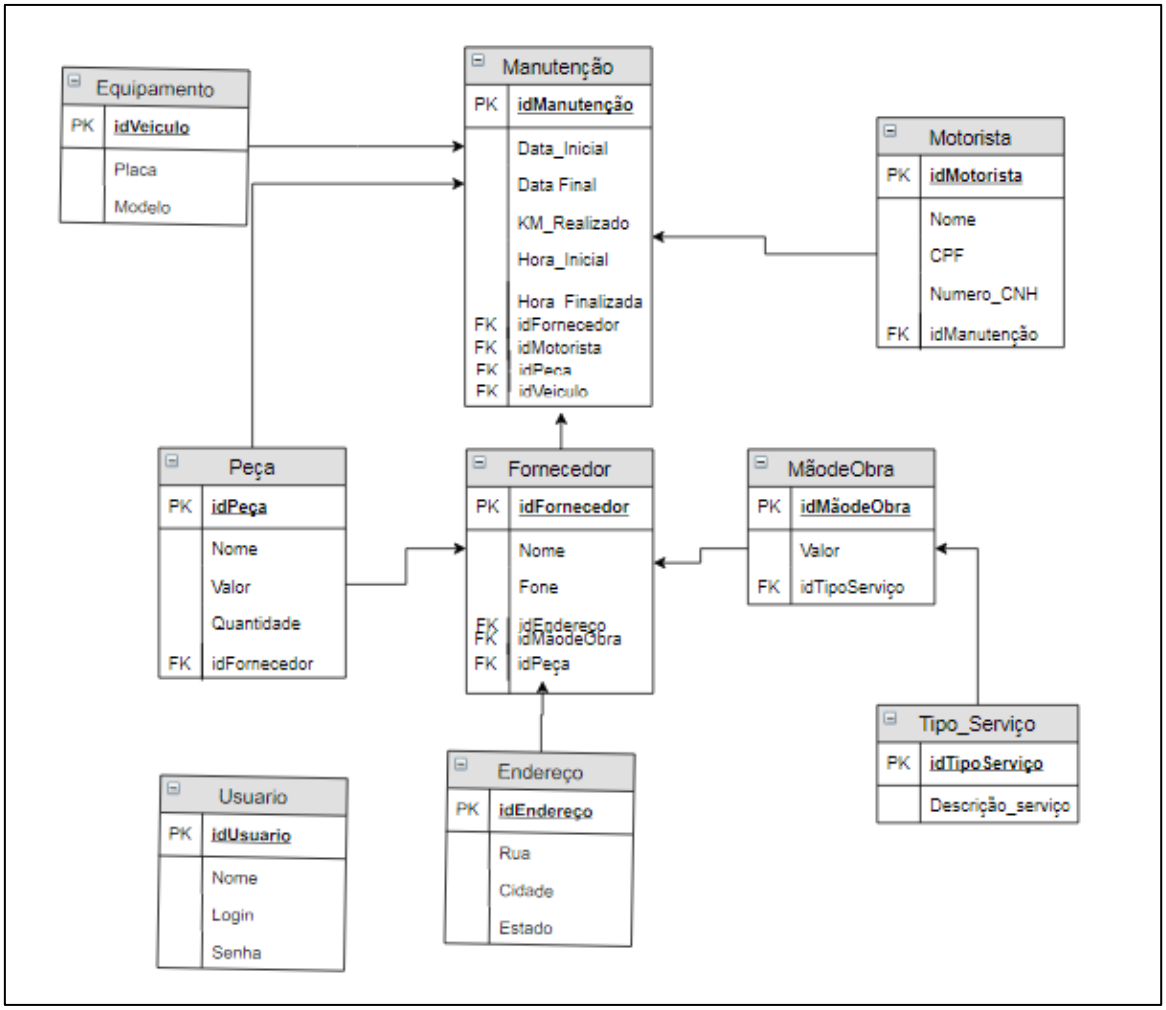

Figura 12: MER aplicado na 3º FN.

Dentro do processo de reestruturação da documentação, não havia nenhuma documentação ou diagrama do sistema antigo. Mas com base nas informações da tela

principal foi desenvolvido a UML, de classes para dar uma melhor noção como ficaria, e funcionaria, após passar por reengenharia. Na Figura 13 podemos ver a UML de classes do sistema de Gestão de Manutenção.

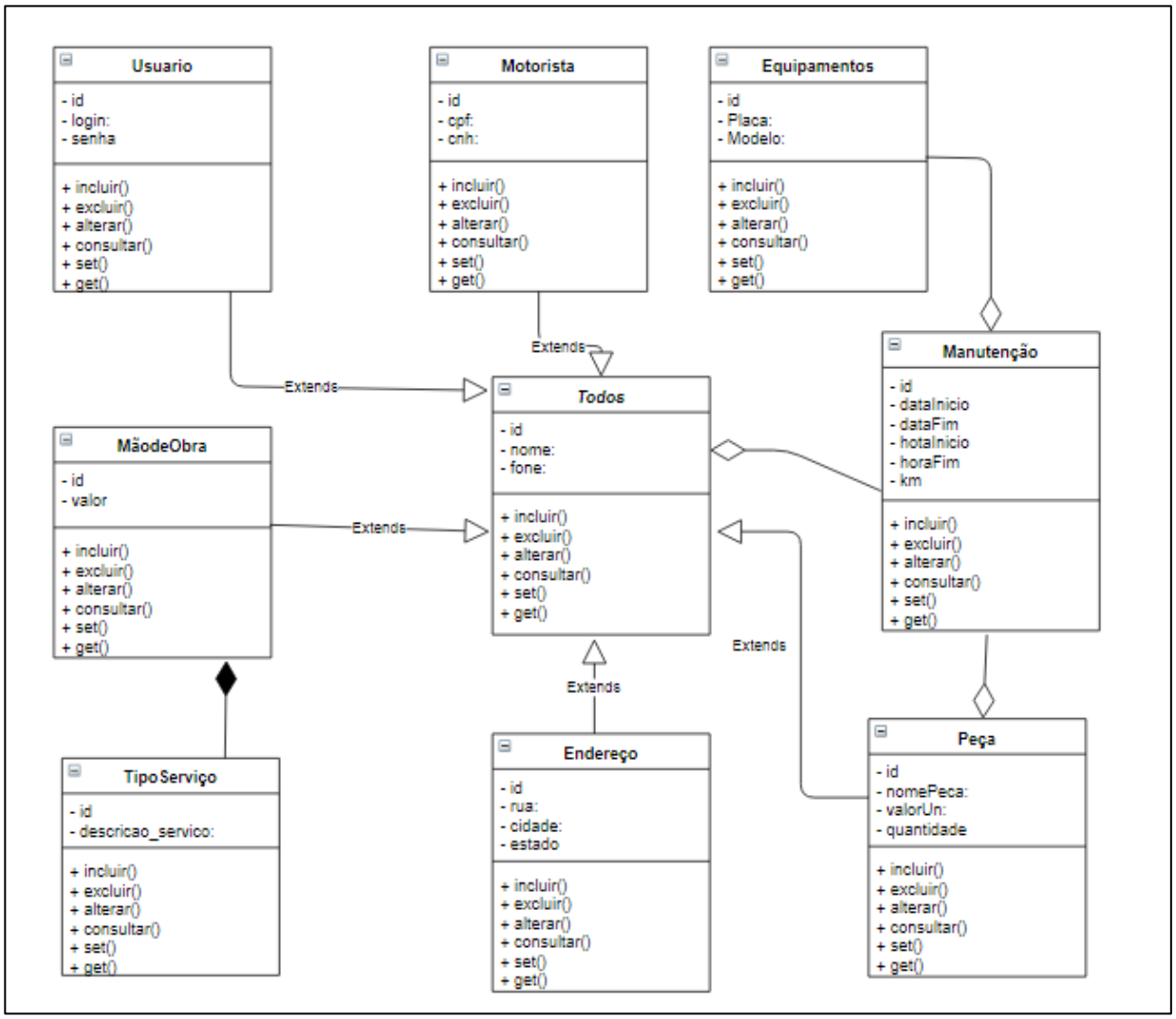

Figura 13: UML, Diagrama de Classes, sistema Gestão de Manutenção.

Como podemos ver no decorrer do trabalho de conclusão de curso, é nítido que o *software* sofreu diversas melhorias, tanto em seu aspecto gráfico, quanto em suas funções operacionais, mesmo não sofrendo alterações de funcionalidade, o mesmo ficou mais dinâmico tendo em vista que com as configurações sugeridas ele pode ser acessado de qualquer computador que possua conexão com a internet, por ser um sistema *web* não se faz necessário a instalação em um *desktop*, tendo um espaço de armazenamento muito maior e confiável, e ligações em seu banco de dados que traz

muito mais confiabilidade, abaixo na Tabela 2, podemos ver os pontos de melhoria e o que essa reengenharia reversa trouxe de positivo para essa aplicação:

COMPARATIVO DE MELHORIAS		
Pontos	Software Legado	Software Após Reengenharia
Liguagem	ACESS 2010	Java
Banco de Dados	ACESS 2010 O banco de dados Microsoft Access Trabalha bem somente para pequenas aplicações com pouca concorrência, capacidade de armazenamento de 2GB.	O banco de dados suporta tabelas de até 32 TB, campos de até 1 GB e linhas de até 1,6 GB ilimitadas. As principais vantagens de usar o PostgreSQL estão relacionadas à economia e ao alto desempenho oferecido pelo SGBD.
Padronização da aplicação	Inesistente	UML, Modelagem Banco de Dados
Aceso	Aplicação desktop é aquela acessada diretamente pelo sistema operacional (em um desktop)	Aplicação web acessada pelo navegador, de qualquer lugar que possua acesso a internet.
Segurança BD	ACESS 2010 esta limitado a um tamanho de 2 GB e não possui um sistema de segurança 100% confiável	PostgreSQL é tão seguro e robusto quanto os BDs em uso e o melhor é que uma implementação dessa pode gerar uma redução de até 75% nos custos de licenciamento e suporte.

Tabela 2: Comparativo de Melhoria

8 - CONSIDERAÇÕES FINAIS

Quando realizamos o processo de reengenharia de *software*, ele tem como base o aproveitamento da lógica aplicada no sistema obsoleto, observando que mesmo com a tecnologia ultrapassada esse sistema ainda tem sua usabilidade. Conseguindo aproveitar o conhecimento agregado, sendo essa abordagem mais adequada do que ir pela opção de iniciar o mesmo do absoluto zero.

Como podemos ver no decorrer desse estudo de caso, foi possível extrair as informações do sistema existente, que nos possibilitou ter uma visão mais clara, conseguindo compreender a aplicação, analisando o que seria útil para utilizar na reengenharia.

Depois da realização da reengenharia conseguimos entender que esse processo não é só com foco, em dar uma cara mais atual para o sistema e sim trazendo atualizações de tecnologia, usabilidade, adaptando suas novas necessidades.

Vimos que esse trabalho nos mostrou apenas o projeto de Reengenharia de Software Legado, trazendo como instrumento de aplicação o SGM, Sistema de Gestão de Manutenção. Concluindo assim a etapa, uma vez que conseguimos demonstrar nos exemplos descritos que a evolução que esse projeto trará é satisfatória. Onde conseguimos ver todo planejamento estruturado dentro da modelagem UML e Banco de Dados, mostrando que o comparativo de melhoria chegou ao que se tinha de objetivo.

9 - REFERÊNCIAS BIBLIOGRÁFICAS

BEZERRA, E. Princípios de Análise e Projeto de Sistemas com UML. 3ª. Edição. Editora Campus, Rio de Janeiro, 2015

DATE, C.J.. Introdução a Sistemas de Banco de Dados. 7ª ed. Rio de Janeiro, RJ: Campus, 2000.

ELMASRI Ramez, NAVATHE Shamkant. Sistema de Banco de Dados 4ª ed. São Paulo,SP: Pearson, 2005.

MACHADO Felipe, ABREU Maurício. Projeto de Banco de Dados Uma Visão Prática. 11º ed. São Paulo, Érica LTDA, 2004.

PÁDUA.Wilson. Engenharia de Software, Fundamentos, Métodos e Padrões. 03. Ed. Rio de Janeiro, 2010.

ROGERS S. Pressman, BRUCE R. Maxim. Engenharia de Software Uma Abordagem Profissional. 8ª ed. Porto Alegre: Amgh, 2016.

SCHACH Stephen. Engenharia de Software: Os Paradigmas Clássico Orientado a Objetos. 7ª ed. São Paulo, SP: Amgh 2010.

SILBERSCHATZ, Abraham & outros. Sistema de Banco de Dados. São Paulo, Makron Books 1999.

SOMMERVILLE. Ian. Engenharia de Software. 09. Ed. São Paulo, 2011.