Engenharia Reversa de um Sistema PDV

Ademir da Cruz dos Santos, Calebe de Paula Bianchini

Faculdade de Computação e Informática - Universidade Presbiteriana Mackenzie (UPM) - São Paulo, SP – Brazil

ade1000ster@gmail.com, calebe.bianchini@mackenzie.br

Abstract . *This work presents a construction of a process proposal for the evolution of legacy systems, using reverse engineering techniques, producing information that can increase the general knowledge of the legacy system. Creating a roadmap to get the main system resources, where there is no access to your source code and no request, the process consists of an initial legacy system evaluation step from a business and technical perspective to transform a legacy system into a modern system, and needs to capture its design, functionality, and identify applications across different elements, and one of the ways to obtain this information is a dynamic system usage analysis method or an ongoing behavior analysis .*

Resumo. *Este trabalho apresenta a construção de uma proposta de processo para a realização de evolução de sistemas legados, utilizando-se a técnica de engenharia reversa, produzindo informações que possam aumentar o conhecimento geral do sistema legado. Criando um roteiro com o objetivo de obter as principais funcionalidades do sistema, onde não temos acesso ao seu código fonte e nenhuma documentação, o processo é composto de uma etapa inicial de avaliação do sistema legado sob perspectiva técnica e de negócio para assim transformar um sistema legado em um sistema moderno, sendo necessário capturar seu design, suas funcionalidades e identificar relacionamentos entre os diferentes elementos, e uma das maneiras de obter essas informações é através de uma análise de uso dinâmico do sistema, ou seja uma análise de comportamento em tempo de execução.*

1. Introdução

A tecnologia evolui constantemente e alguns sistemas de software precisam evoluir para adaptar as necessidades das organizações. Essas evoluções podem ocorrer devido a mudanças nos requisitos comerciais, técnicos, legislação, sistemas operacionais, necessidade de operar em outras plataformas e etc. Um exemplo é a ampla adoção da computação em nuvem, um número cada vez maior de organizações viu como uma importante estratégia de negócios, obrigando a evoluir seus aplicativos herdados para infraestruturas habilitadas para nuvem. A lei de mudança contínua do Lehman [2] afirma que "os sistemas devem ser continuamente adaptados ou tornar-se progressivamente menos satisfatórios." (LEHMAN, 1980).

O software analisado neste trabalho é um sistema de Ponto De Venda (PDV) utilizado para realizar vendas no varejo como supermercados, padarias, açougues e etc, chamado de PDV Coral. Esse software foi desenvolvido a mais de vinte anos, por questões contratuais não claras entre a empresa e o desenvolvedor do software o seguinte cenário se apresenta atualmente: A empresa a qual trabalho e não tem acesso ao código fonte. O desenvolvimento foi realizado por um único programador contratado na época de maneira informal (sem nenhum tipo de documento).

Ao longo dos anos a empresa foi distribuindo esse software em diversos clientes, existem clientes que possui apenas um PDV, alguns com dezenas e uma rede de supermercados que conta com cinquenta lojas que variam de dez a quarenta PDVs por filial somando aproximadamente novecentos PDVs. O programador já deixou claro que não fornece o código fonte, a parceria entre o desenvolvedor e a empresa foi feita de maneira informal. A empresa não se sente encorajada para uma ação judicial ou outro meio legal para obter direito ao código fonte e documentação (caso exista).

Ao longo desses anos, diversas modificações foram realizadas no PDV coral, algumas devido a mudança nas regras fiscais controladas pelo governo, como novas tributações, ou até mesmo a obrigatoriedade de informar o cpf do cliente no ato da compra, como no caso da nota paulista. Outras mudanças por necessidade do cliente, como validação de estacionamento no próprio PDV, promoções específicas do tipo cliente fidelidade e também modificações para corrigir falhas.

As alterações são realizadas apenas pelo programador que possui o código, o qual cobra o valor que lhe convém, e na maioria das vezes não cumpre com o prazo pré-estabelecido, causando um sério problema entre a empresa e o cliente. Apesar do cenário descrito, o PDV Coral atende de maneira satisfatória os clientes, não existe interesse por parte dos clientes em substituir o sistema existente por outro.

O objetivo deste trabalho é realizar engenharia reversa do software PDV Coral analisando o software para criar uma documentação e reprojetar o sistema. Usando a metodologia de extrair as principais funcionalidades, através de testes em tempo de execução, detalhando o seu comportamento, simulando vários cenários possíveis, gerando assim uma representação robusta do sistema. Com esses testes é possível chegar a uma descrições lógicas do sistema como a identificação de diagramas de caso de uso e de sequência. Após a identificação dos principais requisitos funcionais será possível

aumentar o nível de abstração, transformando essas descrições em novas e melhoradas descrições lógicas do sistema definindo como o comportamento especificado será obtido.

A partir dessas descrições, caso a empresa tenha interesse será possível uma futura reengenharia completa do sistema para uma linguagem moderna, mantendo todas as funcionalidades e toda interface para que o cliente não perceba grandes mudanças e a empresa tenha controle sobre o software.

2. Engenharia Reversa

Algumas invenções foram criadas a partir de uma ideia, alguém pensou e criou algo único que alguém jamais havia pensado, porém outras invenções surgem a partir da análise de outras invenções, desenvolvendo de maneira independente do objeto de estudo. A engenharia reversa é essencial para as pesquisas que objetivam entender e melhorar as tecnologias existentes, pois é possível que concorrentes realize o estudo de um objeto mesmo que este seja protegido por leis de propriedade intelectual. De forma ampla, engenharia reversa é o processo de entender como algo funciona através da análise de sua estrutura, função e operação permitindo conhecer como um produto foi pensado e desenhado mesmo sem possuir o projeto original.

Engenharia reversa pode ser realizada em diferentes áreas e não apenas na tecnologia, qualquer um que estuda algo de forma detalhada com intuito de entender o seu funcionamento e como foi feito está fazendo engenharia reversa. Engenharia reversa pode ser usada para entender como algo funciona, ou como foi feito, mesmo sem possuir a sua documentação e a partir desse entendimento posso alterar ou criar outro objeto com as mesmas características do estudado. Pressman (2011, p. 699) afirma que o termo engenharia reversa tem suas origens no mundo do hardware, uma empresa desmonta um produto de hardware competitivo na tentativa de conhecer os “segredos” de projeto e fabricação do concorrente. Os segredos poderiam ser facilmente entendidos se fosse possível obter as especificações de projeto e fabricação do concorrente. Mas esses documentos são de propriedade privada e não estão disponíveis para a empresa que está fazendo a engenharia reversa.

O resultado da engenharia reversa é a especificação de um produto que é gerada a partir do exame realizado no mesmo. A engenharia reversa nos permite entender o funcionamento de algo, para alterar ou realizar manutenções no objeto de estudo, ou mesmo criar um outro objeto, utilizando de técnicas de reengenharia reversa a partir dos dados coletados. Desde a sua primeira utilização, a engenharia reversa cresceu muito, e hoje possibilita desde o entendimento e melhoria daquilo que se estuda, até a descoberta de segredos industriais e comerciais.

2.1. Diferenças Entre Engenharia Reversa e Reengenharia

Apesar da semelhança entre estas palavras, elas resultam em objetivos totalmente diferentes. Para Chikofsky e Cross (1990), IEEE CS-TCSE (1997) e GT-REG (1998), a reengenharia, também chamada de renovação ou reconstrução, é o estudo de um sistema de software, para recriá lo em uma nova forma.

A engenharia reversa de software é um processo de recuperação de projeto, consistindo em analisar um programa, na tentativa de criar uma representação do mesmo, em um nível de abstração mais alto que o código-fonte (PRESSMAN, 1995). A engenharia reversa é a análise de um sistema gerando assim a sua representação, a

reengenharia vai até a criação de um outro sistema. Na reengenharia um sistema é estudado e criado sua representação, com essa representação, cria-se um novo sistema com outra estrutura mas que funciona como o estudado, não sendo uma cópia deste mas procurando realizar as mesmas funcionalidades. Na reengenharia utiliza-se alguma forma de engenharia reversa, seguida por uma forma de engenharia progressiva ou reestruturação.

Benedusi, define engenharia reversa como um conjunto de teorias, metodologias, e técnicas capazes de suportar a extração e abstração de informações de um software existente, gerando documentos consistentes, através do seu código fonte, ou através da adição de conhecimento e experiência que não podem ser automaticamente reconstruídos a partir do código (BENEDUSI et al., 1992 apud RAMAMOORTHY et al., 1996; CHIKOFSKY e CROSS, 1990).

Sommerville (2011), afirma que a engenharia reversa é usada para analisar um programa e com essa análise se possa abstrair e extrair informações que serão usadas para gerar a documentação da organização e funcionalidades de um determinado sistema.

A reengenharia pode envolver a redocumentação do sistema que é o produto da engenharia reversa, pois colabora com o entendimento do sistema que está sendo melhorado ou recriando, sendo assim, uma pode sim depender da outra, porém tal dependência não é obrigatória, elas são diferente pois criam resultados totalmente distintos conforme mostra a figura 1.

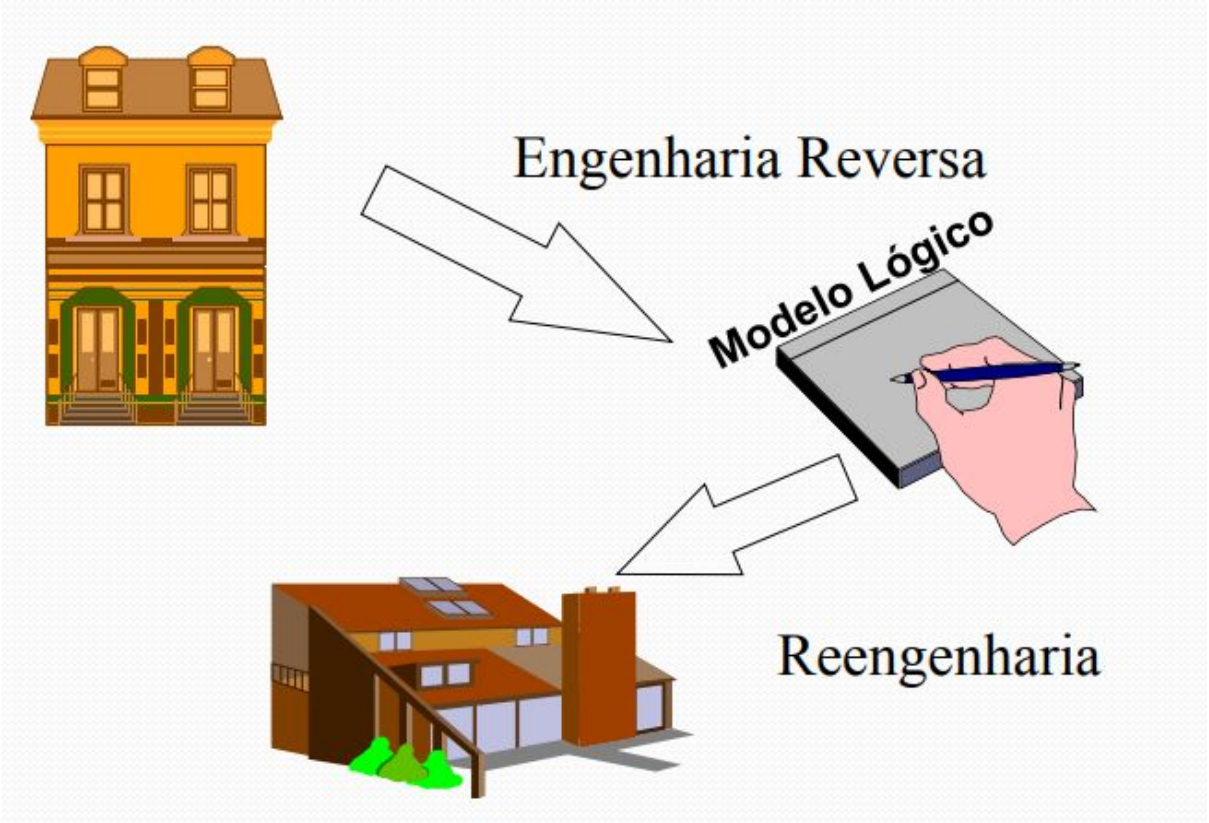

Figura 1 diferença entre engenharia reversa e reengenharia (Yumi, 2015)

2.2. Quando Realizar a Engenharia Reversa?

Cada vez mais os softwares esta fazendo parte do meio em que vivemos, para Pfleeger (2007), nos dias atuais os softwares estão presentes explicitamente ou mesmo sem se fazer notar, em todos os aspectos de nossas vidas, inclusive nos sistemas críticos e complexos que podem afetar nossa saúde e bem-estar, por essa razão, a engenharia de software tornou-se mais importante do que nunca, ressalta ainda que boas práticas de engenharia de software deve assegurar que o software tenha uma boa, grande e positiva contribuição para as vidas dos cidadãos.

A engenharia reversa na área de sistemas começou com a necessidade de compreender o comportamento e detalhes usados na criação de um determinado sistema a partir do estudo do mesmo. O resultado deste estudo poderá resultar em uma documentação com uma visão abstrata dos comportamentos e ações do sistema estudado.

Quando é necessário o uso de Engenharia Reversa na Informática?

Organizações utilizam sistemas que apresentam problemas tais como:

- O sistema é antigo.

- O sistema possui pouca ou nenhuma documentação, ou uma documentação desatualizada que descreve um estado anterior do sistema, mas não o atual.

- Os criadores do sistema já saíram da empresa, e ninguém sabe explicar as decisões que foram tomadas na criação do mesmo.

- Algumas partes do sistema foram alterados sem nenhum controle ou clareza.

- Ao longo do tempo muitos programadores fizeram alterações. Cada um com o seu estilo pessoal de programação.

- O sistema foi criado em uma linguagem de programação antiga.

Porém o sistema precisa evoluir para:

- Para ser adaptado a novos softwares (novas bibliotecas, novas linguagem de programação, novas ferramentas).

- Para ser adaptado a novas regras.

-Para disponibilizar novas funcionalidades.

- Para corrigir bugs.

Todos os sistemas têm um tempo de vida limitado, sendo que cada alteração efetuada pode degenerar a sua estrutura, fazendo com que as manutenções tornem cada vez mais difíceis e dispendiosas. Isso ocorre principalmente em software legado (JACOBSON e LINDSTRÖM, 1991).

2.3. Visões de Software

A partir da engenharia reversa e com base nos diferentes níveis e graus de abstração, o software pode ser visualizado de diferentes maneiras (HARANDI e NING, 1990):

Abstração é a capacidade de se ignorar as características de assuntos não importantes para o propósito em questão, cada etapa no processo de criação do software é um melhoramento do nível de abstração do software, nas etapas iniciais do ciclo de vida as informações possuem alto nível de abstração e nas etapas finais baixo nível de

abstração informações de maneira mais global possuem alto grau de abstração, numa maneira mais detalhada possuem baixo grau de abstração.

- Visão em nível implementacional: abstrai informações da linguagem de programação e características específicas da implementação;

- Visão em nível estrutural: abstrai informações detalhadas da linguagem de programação para mostrar sua estrutura a partir de diferentes perspectivas. O resultado é uma representação explícita das dependências entre os componentes do sistema;

- Visão em nível funcional: abstrai a função de um componente, isto é, o que o componente faz. Essa visão relaciona partes do programa às suas funções, procurando revelar as relações lógicas entre elas (diferentemente das relações sintáticas ou das estruturais);

- Visão em nível de domínio: abstrai o contexto em que o sistema está operando, ou seja, o porquê do sistema a ser desenvolvido.

É relevante ressaltar que uma forma de representação extraída do código pode diferir de uma representação similar que foi desenvolvida no processo de engenharia progressiva. A forma extraída irá refletir a característica comportamental da representação do código muito mais do que a representação original, que reflete a compreensão do problema pelo analista ou projetista.

A Figura 2 mostra a correspondência entre as categorias de visualização do software e as diferentes atividades do ciclo de desenvolvimento de software.

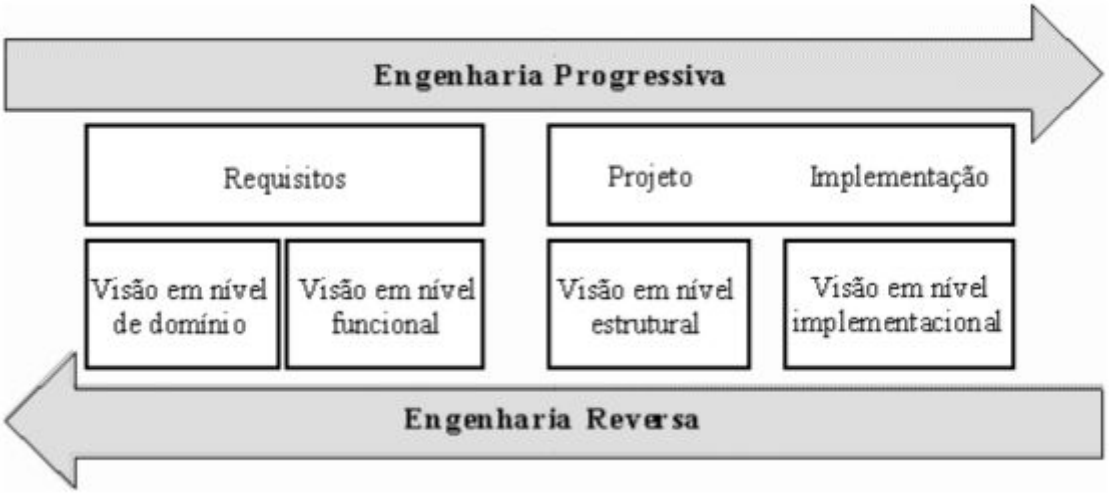

Figura 2: Visualizações de Software no Ciclo de Desenvolvimento (COSTA, 1997).

2.4. Categorias

De acordo com o nível de entendimento obtido do sistema e o escopo das informações usadas, duas categorias de engenharia reversa são definidas: visualização de código (OMAN, 1990) e entendimento de programa (CHIKOFSKY e CROSS, 1990).

2.4.1 Visualização de código.

A visualização de código é a criação ou revisão de representações equivalentes num mesmo nível de abstração (CHIKOFSKY e CROSS, 1990).

O processo de visualização de código permite cria as representações com as informações que foram extraídas na análise do código fonte.

Recuperações mais ambiciosas tais como a função, os propósitos ou a essência do sistema exigem um nível de entendimento maior e são definidas como entendimento de programa.

2.4.2 Entendimento de programa.

Nesta categoria de engenharia reversa, também denominada recuperação de projeto, o conhecimento do domíinio das informações externas e as deduções são adicionadas às observações feitas sobre o sistema através do exame do mesmo, de modo a obter informações com nível mais alto de abstração [CHIKOFSKY e CROSS, 1990).

Segundo Biggerstaff (1989), o entendimento de programa recria abstrações do projeto a partir de uma combinação de código, documentação existente do projeto (se disponível), experiências pessoais e conhecimentos gerais sobre o problema e o domínio de aplicação. Deve produzir todas as informações necessárias para se entender completamente o que, como, e por que o sistema faz.

Entendimento de programa distingue-se de visualização de código porque objetiva entender o sistema, em vez de simplesmente fornecer visões alternativas para auxiliar o usuário a entender o sistema. Esse entendimento vai além do conhecimento em nível implementacional e estrutural, buscando obter o conhecimento em nível funcional e até mesmo em nível de domínio (ambiente de operação do sistema).

Um completo entendimento de um programa busca entender não somente a função do sistema, mas também o processo pelo qual o sistema foi desenvolvido. Rugaber et al. (1990) enfatizam a importância da recuperação de decisões de projeto tomadas durante o desenvolvimento original para uma completa estrutura de entendimento.

3. Engenharia reversa no sistema PDV coral

A funcionalidade principal do PDV coral é realizar vendas, porém existem diversos software no mercado que realiza essa mesma função, de forma diferente, com interfaces diferentes e com requisitos diferentes, informar apenas que o software realiza vendas possui um alto nível de abstração. Como o objetivo é fazer a engenharia reversa, mantendo as funcionalidades do software, sem que o usuário perceba que houve mudança e sem a necessidade de um treinamento, e conforme mencionado anteriormente, não temos acesso ao código fonte nem a documentação do software. É necessário a execução do software e realização da funcionalidade a qual se deseja entender, aumentando assim o nível de abstração conforme vai obtendo os detalhes de como tal funcionalidade é executada.

Antes de iniciar todo o processo de engenharia reversa era necessário ter em mãos o software aqui utilizado no estudo, foi solicitado uma licença e autorização da empresa em que trabalho que é a responsável pela distribuição do PDV Coral, a mesma autorizou e gerou uma licença temporária para testar o software em uma máquina pessoal. Com o

sistema executando iniciamos fazendo um teste da sua principal funcionalidade que é realizar venda.

A funcionalidade de realizar venda pode ser descrita como: “o cliente chega ao ponto de vendas com os produtos da compra, para cada produto o atendente registra o código e a quantidade do produto, o sistema determina o preço do produto e o adiciona a compra, atendente indica que a lista de produtos está completa, o sistema calcula e apresenta o total, o atendente informa o cliente sobre o total da compra e pergunta qual é a forma de pagamento, registra a quantia recebida, o sistema mostra o troco e gera o recibo, a atendente deposita o dinheiro, devolve o troco e entrega o recibo da compra, o sistema registra o final da transação”. Essa descrição possui ainda um alto nível de abstração, para o entendimento do sistema e entender como é realizado cada etapa foi realizada o processo de vendas seguindo o fluxo principal, fornecendo todas as requisições solicitadas pelo sistema. Antes de iniciar a venda o sistema solicita o CPF do cliente, em seguida solicita o código do primeiro produto e ao passar último produto o atendente seleciona finalizar a venda.

Diversos testes foram realizados para identificar os fluxos alternativos do sistema, sabemos que a venda começa com a solicitação do CPF do cliente, no primeiro teste informamos corretamente o CPF e o sistema validou e na próxima etapa solicitou o código do produto seguindo assim seu fluxo principal. Pode ocorrer erro de digitação do CPF ou o cliente não desejar que inclua o CPF, que são os fluxos alternativos, Nos testes seguintes foi possível identificar esses comportamentos, bem como as mensagens de erros ou alerta exibidos pelo sistema conforme o requisito não era informado ou era informado com erros.

O processo descrito no parágrafo anterior foi repetido em todas as etapas, isso é necessário para entender o que, como e porque o sistema faz, através das observações feitas sobre o sistema durante o estudo. O passo seguinte após informar ou não CPF consiste em iniciar o registro dos produtos, seguindo a mesma analisar os diversos cenários possíveis, para identificar possíveis fluxos alternativos. Foi possível identificar como o sistema se comporta quando: o produto não está cadastrado, seu código de barras está ilegível, quais as outras formas de informar para o sistema qual produto a atendente está tentando incluir na compra, (Scanner, busca por descrição, informar o código interno, etc), cliente deseja excluir um produto, cliente deseja incluir um produto(aumentar a quantidade). Esse padrão de análise foi repetido em outras funcionalidades do sistema como: Cancelar venda, Consultar produto, Finalizar venda.

Esses testes realizado no software produziu as informações necessárias para se entender o sistema e suas principais funcionalidades baseado em como o sistema funciona gerando o diagrama de casos de uso figura 3 e suas especificações representando o entendimento do sistema.

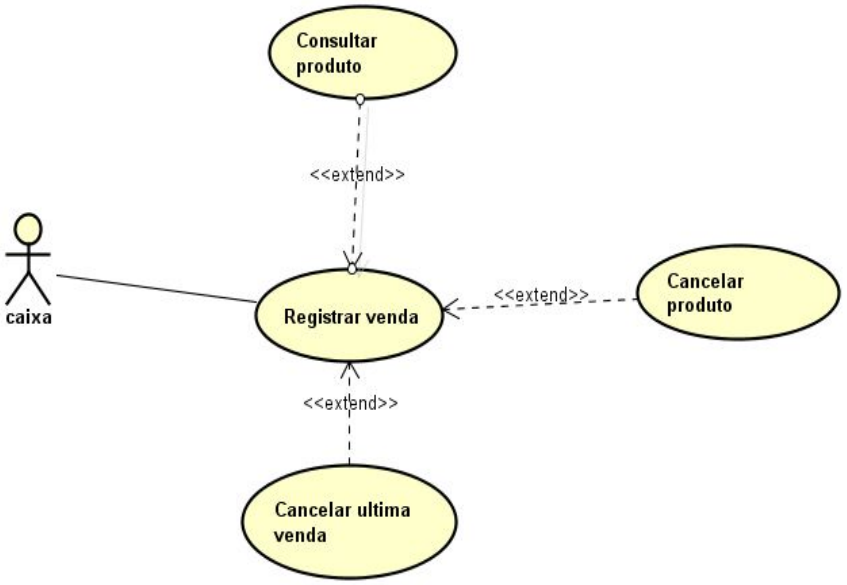

Figura 3: diagrama de caso de uso.

Especificações:

[UC001 - Registar Venda]

Descrição Resumida: Este UC tem como objetivo realizar o processo de venda.

Ator Principal: Caixa (Operador de caixa).

Pré-Condições : Caixa deve está no modo de venda e não como caixa fechado.

Pós-Condição: Sistema finaliza a venda e emite cupom fiscal.

Pontos de extensão: Cancelar produto, passos 4 e 5. Consultar produto, pasos 1 Cancelar Venda Anterior, passo 3.

Gatilho: O operador de caixa deseja realizar o registro de uma venda.

Tipo de Fluxo: Principal

[FP01 - Registrar Venda]

1. Caixa informa o cpf do cliente;
2. Sistema valida o cpf do cliente; **[FA01]**
3. Sistema solicita ean do produto; **[FA02] [FA03]**
4. Caixa informa Ean do produto;

5. Caso seja o primeiro produto sistema efetua a abertura da compra, registra o produto;
6. Sistema exibe a descrição e preço do produto atualiza e exibe o valor parcial da compra;

   Caixa repete os passos 3 a 6 até o ultimo produto da compra;

7. Caixa solicita o valor total;
8. Sistema informa o valor total da compra e solicita o pagamento;
9. Caixa seleciona a forma de pagamento e informa o valor recebido;
10. Sistema exibe valor recebido e troco e finaliza a venda;
11. Sistema emite o comprovante fiscal (Fim do UC);

Tipo de Fluxo: Alternativo

[FA01 - Caixa digita cpf inválido]

1. Sistema informa que o cpf é incorreto;

2 Sistema volta para fluxo principal no passo 1.

Tipo de Fluxo: Alternativo

[FA02 - Venda por quantidade]

1. Caixa informa quantidade e Ean do produto;
2. Sistema volta para fluxo principal no passo 4. Sistema volta para fluxo principal no passo 1.

Tipo de Fluxo: Alternativo

[FA03 - Busca pela descrição do produto]

1. Caixa seleciona o produto pela descrição;
2. Sistema identifica o código do produto;
3. Sistema volta para fluxo principal no passo.

[UC002 - Cancelar Produto]

Descrição Resumida: Este UC tem como objetivo Cancelar um Produto em uma Venda

Ator Principal: Caixa (Operador de caixa).

Pré-Condições: Sistema deve está no modo de venda com ao menos um produto registrado.

Pós-Condição: Sistema Cancela o item e continua o processo de vendas.

Pontos de extensão: Inexistente.

Gatilho: Caixa deseja cancelar um produto, registrado na venda.

Tipo de Fluxo: Principal

[FP01 - Cancelar Produto]

1. Caixa solicita cancelar um produto;
2. Sistema solicita autorização do supervisor;
3. Caixa informa a autorização do supervisor;
4. sistema valida o supervisor; **[FA01]**
5. Sistema solicita ean do produto; **[FA02]**
6. Caixa informa Ean do produto; **[FA03]**
7 Sistema realiza o cancelamento do produto, exibe a descrição e preço do produto cancelado e atualiza o valor parcial da compra. (Fim do UC).

Tipo de Fluxo: Alternativo

[FA01 - Supervisor não identificado]

1. Sistema informa que o supervisor não existe;
2. Sistema volta para fluxo principal no passo 2.

Tipo de Fluxo: Alternativo

[FA02 - Cancelar por quantidade]

1. Caixa informa quantidade e Ean do produto;
2. Sistema volta para fluxo principal no passo 6.

Tipo de Fluxo: Alternativo

[FA02 - Caixa informa produto que não está registrado na venda]

1. Sistema informa que o cancelamento não pode ser realizado;
2 Sistema volta para fluxo principal no passo 6.

[UC003 - Consultar Produto]

Descrição Resumida: Este UC tem como objetivo consultar o preço de um produto.

Ator Principal: Caixa (Operador de caixa).

Pré-Condições: Inexistente.

Pós-Condição: Sistema informa na tela o valor e descrição do produto consultado.

Pontos de extensão: Inexistente.

Gatilho: Caixa solicita consultar um produto.

Tipo de Fluxo: Principal

[FP01 - Consultar Produto]

1. Caixa solicita consultar um produto;
2. Sistema solicita ean do produto; **[FA01][FA02]**
3. Caixa informa Ean do produto;
4. Sistema informa descrição e valor do produto consultado. (Fim do UC).

Tipo de Fluxo: Alternativo

[FA01 - Caixa Informa produto inexistente]

1. Sistema informa que o produto não existe;
2. Sistema volta para fluxo principal no passo 1.

Tipo de Fluxo: Alternativo

[FA02 - Busca pela descrição do produto]

1. Caixa seleciona o produto pela descrição;
2. Sistema identifica o código do produto;
3. Sistema volta para fluxo principal no passo 3.

[UC004 - Cancelar Última Venda]

Descrição Resumida: Este UC tem como objetivo Cancelar Venda.

Ator Principal: Caixa (Operador de caixa).

Pré-Condições: Sistema deve está no modo de venda sem produto registrado.

Pós-Condição: Sistema Cancela a compra, exibe mensagem e emite cupom de cancelamento.

Pontos de extensão: Inexistente.

Gatilho: Caixa deseja cancelar a última venda.

Tipo de Fluxo: Principal

[FP01 - Cancelar Última Venda]

1. Caixa solicita cancelar a venda anterior;
2. Sistema solicita autorização do supervisor;

3. Caixa informa a autorização do supervisor;

4. sistema valida o supervisor;**[FA01]**

5. Sistema realiza o cancelamento e exibe mensagem;**[FA02]**

6. Sistema emite cupom de cancelamento (Fim do UC).

Tipo de Fluxo: Alternativo

[FA01 - Supervisor não identificado]

1. Sistema informa que o supervisor não existe;

2. Sistema volta para fluxo principal no passo 3.

Tipo de Fluxo: Alternativo

[FA02 - Venda anterior não encontrada]

1. Sistema informa que o cancelamento não pode ser realizado;

2. Sistema volta para fluxo principal no passo 1.

Com as informações extraídas no processo de engenharia reversa é possível iniciar o processo de Reengenharia, identificando quais classes poderia utilizar para desenvolver esse software mantendo as mesmas funcionalidades, iniciando assim a elaboração do modelo de domínio.

A figura 4 mostra um diagrama modelo de domínio proposto a partir da análise da documentação resultado do processo de engenharia reversa.

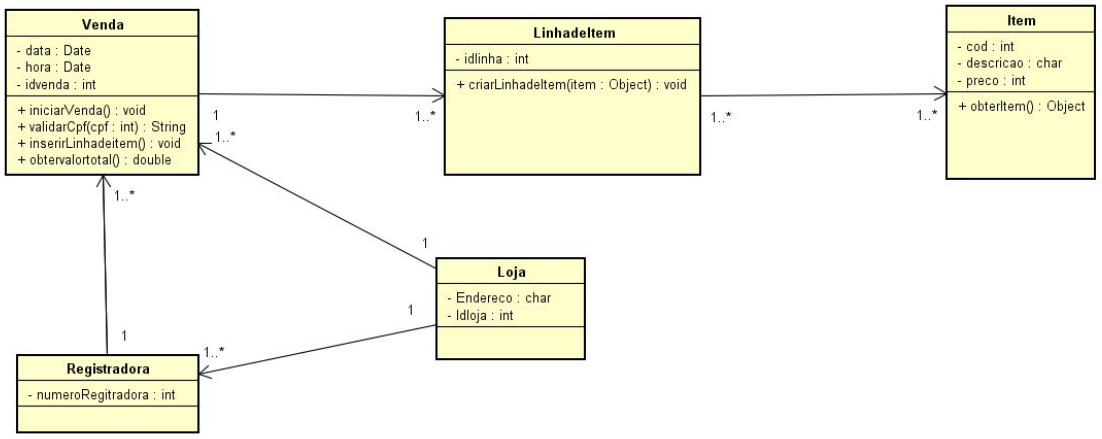

Figura 4. diagrama modelo de domínio.

4. Conclusão

A engenharia reversa permitiu extrair informações do software legado PDV Coral, fornecendo visões abstratas do mesmo, permitindo compreender o software e gerar documentação que poderão ser utilizados na reengenharia. Apesar da falta de

documentação e sem acesso ao código fonte, foi possível realizar engenharia reversa através da categoria de entendimento do programa. Os testes realizados no software PDV Coral executando e se passando como usuário, fazendo todas as operações, simulando diversos cenários possíveis, permitiu entender suas funcionalidades, que geraram os diagramas de casos de uso e de seqüencia.

O software aqui estudado foi autorizado pela empresa proprietária, que forneceu a licença e a devida autorização, Porém tal estudo seria possível mesmo sem o conhecimento do proprietário, um concorrente adquire uma licença e estuda o comportamento desse software para entender uma determinada funcionalidade e implementar em um outro sistema.

A falta de documentação e acesso ao código fonte, dificulta o entendimento do software, com uma documentação em mãos e análise de algumas funções no código isso tornaria bem mas fácil, porém esse é o objetivo aqui, recuperar o projeto, gerar uma documentação para auxiliar a uma possível reengenharia que criaria um novo software com as mesmas características justamente por que o software legado não tem seu código fonte e documentação.

5. Referências Bibliográficas

BENEDUSI, P.; CIMITILE, A.; CARLINI, U. Reverse Engineering Processes, Design Document Production, and Structure Charts. Journal Systems and Software, v. 19, 1992, p. 225-245.

BIGGERSTAFF, T. J. Design Recovery for Maintenance and Reuse. IEEE Computer, v. 22, n. 7, 1989, p. 36-49.

Chikofsky, E.J., CrossII, J.H., 1990. Reverse engineering and design recovery: A taxonomy. IEEE Software 7 (1), 13–17.

COSTA, R. M. Aplicação do Método de Engenharia Reversa FUSION-RE/I na Recuperação da Funcionalidade da Ferramenta de Teste PROTEUM. São Carlos: ICMSC-USP, 1997. Dissertação (mestrado).

GT-REG (Georgia Tech's - Reverse Engineering Group). Glossary of Reengineering Terms. Georgia, http://www.cc.gatech.edu/reverse/glossary.html, 1998.

HARANDI, M. T.; NING, J. Q. Knowledge-Based Program Analysis. IEEE Software v. 7, n. 1, 1990, p. 74-81.

IEEE CS-TCSE (IEEE Computer Society - Technical Council on Software Engineering). Reengineering & Reverse Engineering Terminology. Washington, http://www.tcse.org/revengr/taxonomy.html, 1997.

JACOBSON, I.; LINDSTRÖM, F. Re-engineering of old systems to an object-oriented architecture. SIGPLAN Notices, v. 26, n. 11, 1991, p. 340-350.

LEHMAN, M. M. Programs, Life-Cycles, and the Laws of program Evolution. Proc. IEEE, 1980, p. 1060-1076.

PFLEEGER, S. L. Engenharia de Software: teoria e prática, 2° ed. São Paulo, 2007, PEARSON.

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

PRESSMAN, R. S. Engenharia de Software. São Paulo: Makron Books, 1995.

RAMAMOORTHY, C. V.; TSAI, W. Advances in Software Engineering. IEEE Computer, v. 29, n. 10, 1996, p. 47-58.

RUGABER, S. et al. Recognizing Design Decision in Programs. IEEE Software, v. 7, n. 1, 1990, p. 46-54.

SOMMERVILLE, I. Engenharia de Software, 9° ed. São Paulo, 2011, PEARSON.

Yumi, E. N. Engenharia Reversa e Reengenharia, 2015.