UNIVERSIDADE DO VALE DO RIO DOS SINOS – UNISINOS
UNIDADE ACADÊMICA DE EDUCAÇÃO ONLINE
ESPECIALIZAÇÃO EM ENGENHARIA DE SOFTWARE

Giovani Bocchese Pereira

Aplicação de Reengenharia em Software Industrial

Porto Alegre
2019

UNIVERSIDADE DO VALE DO RIO DOS SINOS – UNISINOS
UNIDADE ACADÊMICA DE EDUCAÇÃO ONLINE
ESPECIALIZAÇÃO EM ENGENHARIA DE SOFTWARE

Giovani Bocchese Pereira

Aplicação de Reengenharia em Software Industrial

Trabalho de Conclusão de Curso apresentado como requisito parcial para a obtenção do título de Especialista em Engenharia de Software, pelo curso de Pós-Graduação Lato Sensu em Engenharia de Software da Universidade do Vale do Rio dos Sinos – UNISINOS.
Orientador: Prof. Msc. Márcio Miguel Gomes

Porto Alegre
2019

Aplicação de Reengenharia em Software Industrial

Giovani Bocchese Pereira[1]

[1]Unidade acadêmica de Educação Online
Universidade do Vale do Rio dos Sinos (UNISINOS) São Leopoldo – RS – Brasil

bp.giovani@gmail.com

Abstract. *Software reengineering aims to rebuild a system that is difficult to maintain but is indispensable in the organization that operates. The objective of the present work was the application of the reengineering process in legacy software for industrial use. This software is a tool for programming electronic products, which due to poor design and several poorly done maintenance, has presented a rapid degradation and difficulty to evolve to meet the new yearning of the company that uses it. This project was done by a bibliographic study regarding reengineering followed by an action-research, with the proper application of the methods studied. With the data obtained it was possible to reimplement it in a new way, meeting the expectations of the company and delivering new documentation that facilitates the maintenance.*

Resumo. *A reengenharia de software visa reconstruir um sistema que está com dificuldades de ser mantido mas que é indispensável na organização que atua. O presente trabalho teve por objetivo a realização do processo de reengenharia em um software legado de uso industrial. O sistema é um gravador de produtos eletrônicos, que devido a má execução de projeto e várias manutenções mal feitas, apresentou uma rápida degradação dificultando a evolução para atender novos anseios da empresa que o utiliza. A atividade consistiu em realizar um estudo bibliográfico a respeito de reengenharia e uma pesquisa-ação, com a devida aplicação dos métodos estudados. Com os dados obtidos foi possível realizar sua reimplementação em uma nova forma, atendo as expectativas da empresa e entregando uma nova documentação que facilita a manutenção.*

1. Introdução

Atualmente podemos afirmar que muitos dos produtos e serviços utilizados pelas pessoas no dia a dia não existiriam sem softwares. Muitas dessas pessoas não tem noção de quão complexo um software em uso no seu cotidiano pode ser. Segundo Sommerville (2011), está complexidade é atrelada ao fato que um software não é governado por leis da física ou restritos por barreiras impostas por fabricação ou materiais utilizados em sua construção. Software é algo abstrato, ele é produto não concreto de uma ideia e que não podemos perceber sua existência através dos nossos sentidos.

Um aspecto que aumenta a complexidade e que entra em conflito com a ideia de algo abstrato, é a de que um software pode envelhecer tal qual um ser humano. Este fenômeno vem ganhando uma importância cada vez maior devido ao crescente uso de software. Há dois fatores que influenciam neste envelhecimento. Um deles é a inabilidade da empresa responsável pelo desenvolvimento perceber as mudanças necessárias para o

sistema e o outro é a consequência da aplicação de mudanças ou manutenções malfeitas, que podem inserir falhas no sistema e até piorar a condição do código fonte (PARNAS, 1994). Segundo Osborne e Chikofsky (1990), um programa pode ter sido criado nas melhores técnicas de desenvolvimento e arquitetura, mas se as alterações e manutenções forem feitas por programadores sem muito conhecimento do programa como um todo, o sistema irá entrar em estado de degradação.

Ao atingir o ponto que a manutenção é custosa e de difícil execução, surge a necessidade de uma reconstrução do sistema, com o processo chamado de reengenharia de software. Com acesso ao sistema existente através do código fonte, busca-se abstrair as funcionalidades e com isso construir o modelo de análise e projeto do software (PIEKARSKI; QUINAIA, 2000).

Segundo Sommerville (2011, p. 174) existem dois benefícios importantes na aplicação de reengenharia ao invés de total substituição por um novo desenvolvimento:

- Risco reduzido: partindo de um software já implementado, reduz-se a chance de erros na especificação do software, principalmente para aqueles de alto risco.
- Custo reduzido: se comparado com o desenvolvimento de um novo software, o custo de reengenharia pode ser menor.

Este trabalho propõem evoluir um software através da aplicação de reengenharia de software e avaliação de forma qualitativa se o sistema está de acordo com o legado. O sistema em questão é um Gravador de Produtos instalado no setor de produção de uma empresa de equipamentos eletrônicos, chamada Comlink. Ele é utilizado para automatizar o processo de seleção de firmware para ser gravado em um produto eletrônico e inserir número de série dentro do programa. O sistema é vital para o setor de produção da empresa, pois facilita o acesso dos operadores aos instrumentos de gravação e seleção correta dos firmwares. Entretanto o sistema não foi documentado e as alterações feitas nos últimos tempos para comportar novas funcionalidades deixaram o código em uma situação de difícil manutenção. Com isto, busca-se resgatar através da reengenharia os requisitos do software e modelar um novo sistema com as mesmas funcionalidades, porém com o código padronizado, documentado e de acordo com as novas expectativas da empresa.

Com a aplicação da reengenharia busca-se responder a seguinte pergunta: a reengenharia é uma técnica adequada para aplicar na evolução do Gravador de Produtos da Comlink? Para tal, foram traçados os seguintes objetivos específicos:

- Aplicar reengenharia com base na bibliografia estudada;
- Implementar o sistema com base na modelagem obtida na reengenharia;
- Validar o sistema no ambiente de aplicação;

O desenvolvimento do trabalho está divido em três partes. A primeira parte deste trabalho contempla o estudo bibliográfico realizado para construir a base de conhecimento necessária ao desenvolvimento da proposta e uma apresentação dos trabalhos relacionados ao tema de evolução e reengenharia de software.
A segunda parte consiste no desenvolvimento do sistema. Iniciando pela aplicação de reengenharia no software legado e após a modelagem e desenvolvimento do novo software. A terceira e última apresenta o encerramento da atividade com as conclusões obtidas através da implementação do software no seu ambiente de trabalho.

2. Fundamentos teóricos

Neste capítulo encontra-se o estudo dos principais assuntos abordados neste trabalho. O objetivo do estudo é buscar um aprofundamento e também referências que servem de guia para o desenvolvimento da solução.

2.1. Reengenharia de software

Como já citado na introdução deste trabalho, o uso de softwares no nosso cotidiano é cada vez maior, difícil encontrar algo que não esteja vinculado a algum sistema computadorizado. Dentro dessa miríade de softwares em uso, temos vários sistemas considerados legados, que são aqueles mais antigos desenvolvidos a pelo menos 10 anos atrás. Osborne e Chikofsky (1990) citaram que em 1990 a maioria dos sistemas em uso na época tinham uma idade média entre 10 a 15 anos, e que mesmo desenvolvidos dentro das melhores práticas de codificação e estruturação de software, foram pensados para ambientes com recursos computacionais restritos. Com o passar do tempo e para manter-se útil a sua organização, um software legado precisa evoluir. Pressman (2016) cita as quatro principais razões que um sistema legado passe por um processo de evolução:

- Necessidade de atender novas tecnologias da computação;
- Implementação de novos requisitos do negócio;
- Troca de tecnologia de banco de dados
- Reestruturar arquitetura para funcionar em um novo ambiente;

Segundo Sommerville (2004), muitas organizações buscam evoluir seus sistemas quando elas desejam aumentar a vida útil de seus softwares, através de uma manutenção mais fácil. Isto é possível como uma nova documentação, mudança na linguagem de programação e reestruturação de dados e do programa. Surgindo a necessidade de evoluir um sistema, dadas as razões acima, temos várias alternativas para serem usadas, e neste trabalho o caminho escolhido foi através da reengenharia de software.

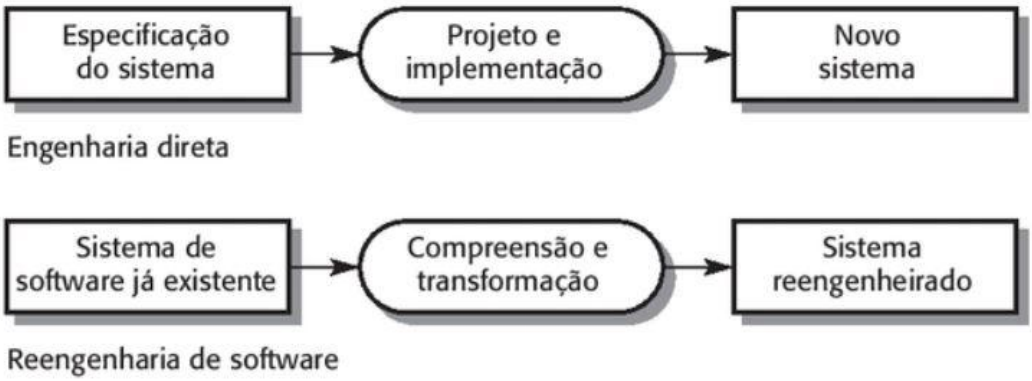
A motivação principal para o uso de reengenharia ao invés de um novo desenvolvimento de software é a diminuição dos riscos e custos que envolvem desenvolver um novo sistema. Por vezes podem ocorrer erros na especificação, devido a esquecimento ou até falta de documentação do sistema legado (SOMMERVILLE, 2011). Warren e Ransom (2002) ressaltam que o uso de reengenharia é uma boa saída para várias organizações, pois o custo de um novo desenvolvimento e implantação pode ser muito alto, tornando-o inviável (apud CAGNIN, 2005). Entretanto, Rosenberg (1996) cita que mesmo a reengenharia sendo empregada para mitigar riscos, não significa que não seja necessário fazer um trabalho para identificação dos riscos do processo de reengenharia. A autora cita alguns:

- Deriva dos esforços de reengenharia;
- Falta de métricas para avaliação;
- Reengenharia sem participação de *experts* do negócio;
- Sistema reimplantado com problemas;
- Dificuldade de abstrair informações do código devido a linguagem de programação;

Conceitualmente, a reengenharia pode ser descrita como: “(...) é a análise e alteração de um sistema para reconstitui-lo em uma nova forma e subsequente

implementação da nova forma" (CHIKOFSKY; CROSS, 1990). O objetivo básico da reengenharia é buscar um completo entendimento do software e de suas camadas de abstração (especificação, projeto e implementação) para reimplementa-lo, buscando manter as mesmas funcionalidades, o deixar pronto para novas funções e ainda facilitar futuras manutenções. Se comparado com o desenvolvimento de um novo software, a reengenharia se diferencia no ponto de partida do desenvolvimento. Ela parte do sistema já implementado, iniciando a análise nas interfaces de usuário, se existentes, e avançando para o código fonte. Já um novo desenvolvimento, parte dos requisitos de software. A figura 1 mostra a comparação do fluxo de trabalho de ambos métodos (ROSENBERG, 1996) (SOMMERVILLE, 2004).

Figura 1: Comparação entre processos



Fonte: (SOMMERVILLE, 2004)

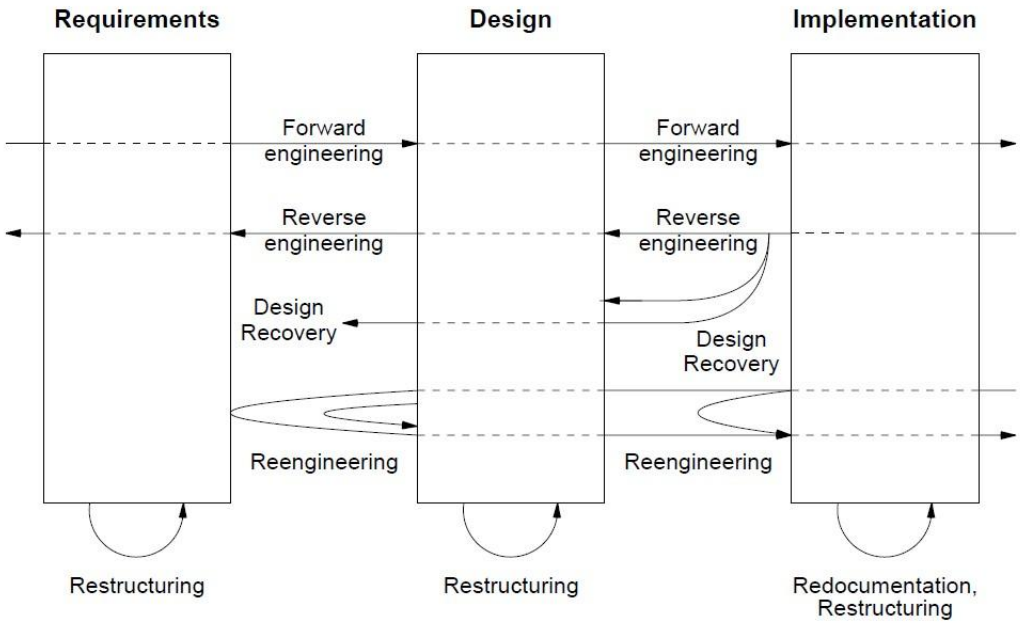
Na figura 2 podemos observar que a reengenharia engloba o uso de duas tecnologias utilizadas para implementar e examinar um software: a engenharia-reversa e a engenharia-avante. Estas duas tecnologias são independentes uma da outra e não podem ser consideradas como algo que pertence somente a reengenharia. Ambas tem seus campos de atuação independentes, a reengenharia apenas as usa para cumprir com o seu objetivo básico. As técnicas de engenharia reversa ajudam no entendimento do sistema, podendo partir da camada de implementação para chegarmos ao projeto e partir do projeto para chegarmos aos requisitos propostos inicialmente. Na direção oposta, temos a engenharia-avante, que trata das técnicas básicas de desenvolvimento de um software. A reengenharia a utiliza para reimplementar o sistema reconstituído (CHIKOFSKY; CROSS, 1990).

A figura 2 mostra também outros termos que não possuem relação direta com o processo de reengenharia, mas acabam sendo produtos indiretos do processo devido a necessidade de alteração do sistema para constituir nova forma. São eles:

- Reestruturação: é classificada como modificação de software para deixar menos suscetível a erros em manutenções futura. Entretanto, este processo ocorre dentro da mesma camada de abstração, pois o foco da reestruturação é melhoria da estrutura de código e não envolve nenhuma modificação (ARNOLD, 1989) (CHIKOFSKY; CROSS, 1990).

- Recuperação de projeto: É a combinação de código, conhecimento do domínio de aplicação do software, documentação, experiências pessoais e conhecimentos gerais. Todo este conhecimento deve ser disponibilizado para uma pessoa entender o que o programa faz, porque ele faz e como ele faz. Este processo é uma parte da engenharia-reversa (CHIKOFSKY; CROSS, 1990).
- Redocumentação: É uma parte do processo de engenharia-reversa, onde o foco é buscar toda documentação do sistema e retrabalha-la para facilitar o entendimento do software. Ela ocorre dentro da mesma camada de abstração do sistema (CHIKOFSKY; CROSS, 1990).

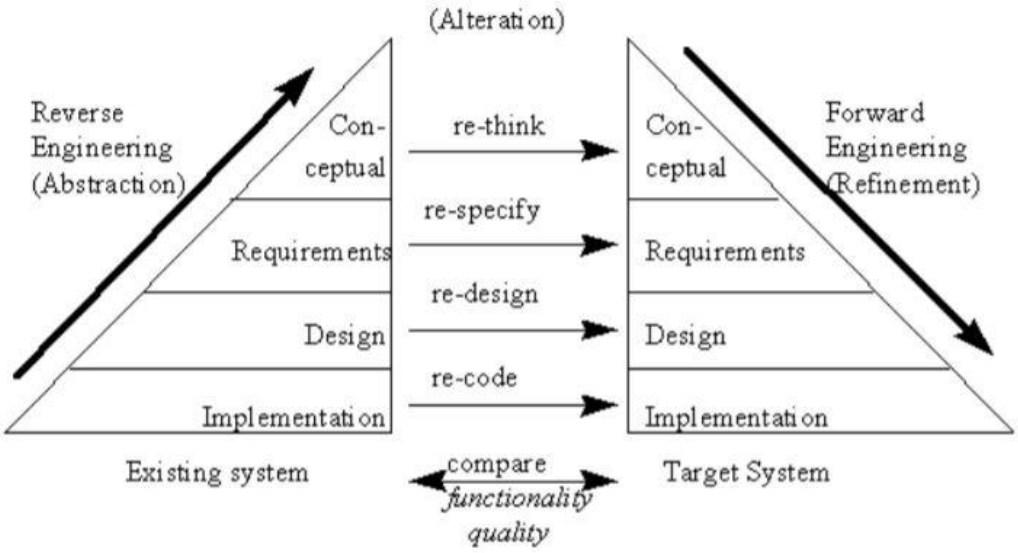
Figura 2: Relação entre termos da reengenharia



Fonte: (CHIKOFSKY; CROSS, 1990)

Rosenberg (1996) apresenta um modelo de reengenharia mais aprofundado na figura 3. Nele a autora acrescenta uma camada de abstração a mais no processo, chamada de camada conceitual, presente no topo da pirâmide. Podemos observar que o processo de engenharia reversa avança da base para o topo, ou seja, aumentando o nível de abstração do sistema existente. Segundo Rosenberg (1996), " Abstração produz a representação que enfatiza certas características do sistema enquanto esconde outras". Desta maneira partindo da implementação e aplicando técnicas de engenharia reversa, podemos chegar no modelo conceitual do sistema, onde toda a ideia de requisitos para ele tem surgido. Sem alterar o nível de abstração, pode-se fazer alterações na camada. Na parte conceitual podemos repensar o sistema, já na camada de requisitos pode-se retrabalha-los adicionando ou apagando requisitos. O processo de engenharia-avante neste modelo, representa a descida do nível de abstração, ou seja, o modelo passa a se tornar real, com as devidas alterações em suas camadas de abstração. Note-se que ao pé da pirâmide consta o processo de comparação, para verificar se a funcionalidade manteve-se a mesma e se a qualidade do sistema melhorou.

Figura 3: Modelo geral de reengenharia



Fonte: (ROSENBERG, 1996)

Sobre como conduzir um processo de reengenharia, várias maneiras foram encontradas na bibliografia estudada. Rosenberg (1996) apresenta três aproximações que podem ser utilizadas para conduzir um processo de reengenharia, cada um com suas vantagens e riscos. A primeira aproximação é chamada de *Big Bang* e consiste em uma substituição total do sistema de uma só vez. Ideal para aplicações que necessitem uma solução imediata para algum problema. Tal solução, entretanto, traz riscos altos, pois nem sempre o sistema reimplementando pode funcionar corretamente de início, principalmente tratando-se de um software de grandes proporções.

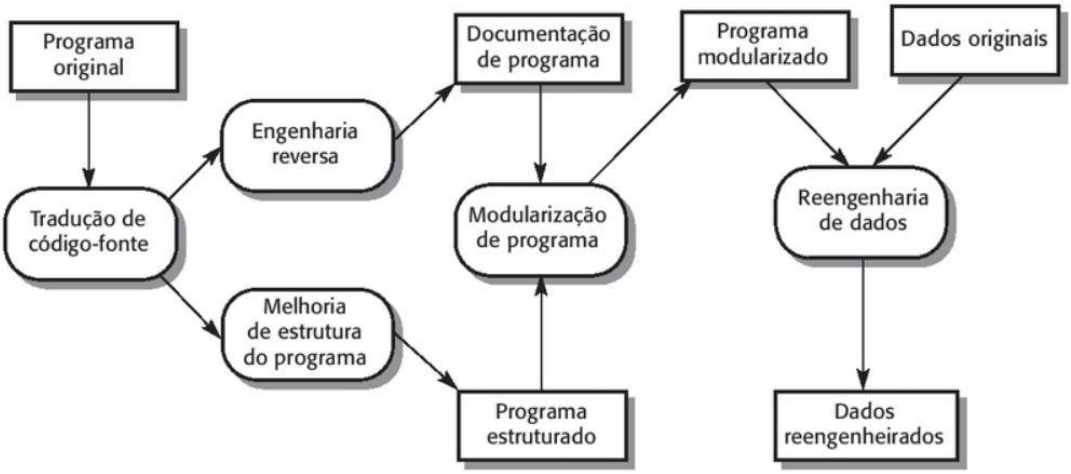
A segunda aproximação sugerida é chamada de incremental. O sistema alvo de reengenharia é quebrado em partes ou blocos baseados na estrutura do sistema, onde em cada parte é realizada reengenharia. Ao finalizar um bloco, ela é entregue como uma versão do software alvo para ser usada e testada. Esse fluxo de entregas garante que cada parte seja testada e avaliada se não perdeu nenhuma funcionalidade. A desvantagem desse método é o tempo de desenvolvimento que pode se estender e é necessário um bom planejamento ao fazer a divisão em blocos. A reengenharia deve ser confinada a estrutura do bloco em trabalho, não podendo afetar outros blocos, assim evitando problemas de compatibilidade.

A última aproximação sugerida por Rosenberg (1996) é chamada de evolucionária. Diferente da incremental, que é divida em partes estruturais, a evolucionária é dividida em partes baseadas na funcionalidade. O software alvo é quebrado em funções, onde cada função isoladamente passa por reengenharia. O resultado desse processo é um sistema altamente modular, com escopo de componentes bem definidos. A desvantagem é na parte que envolve funcionalidades semelhantes, que deverão ser unificadas para evitar problemas de interface.

Dadas as aproximações possíveis para conduzir um processo de reengenharia,

deve-se também estudar as etapas necessárias. Sommerville (2004) propõem um diagrama de fluxo, mostrado na figura 4. A ideia deste processo é gerar uma versão remodularizada de um programa legado. O autor salienta que nem sempre as etapas mostradas são necessárias executar, como a tradução de código fonte e a reengenharia de dados.

Figura 4: Modelo básico de reengenharia



Fonte: (SOMMERVILLE, 2004)

2.2. Engenharia reversa

Engenharia reversa é um termo que causa bastante confusão quanto ao seu conceito e uso (CHIKOFSKY; CROSS, 1990). Como já citado na seção de reengenharia, a engenharia reversa é uma ferramenta utilizada dentro da reengenharia. Ela tem sua própria linha de evolução e é empregada em outras técnicas de evolução e manutenção de software. Engenharia reversa nasceu na área de hardware, onde é muito comum seu uso para decifrar projetos de produtos finalizados, com objetivo de entender o concorrente e até mesmo clonar o projeto para utilizar em seus próprios produtos (CHIKOFSKY; CROSS, 1990).

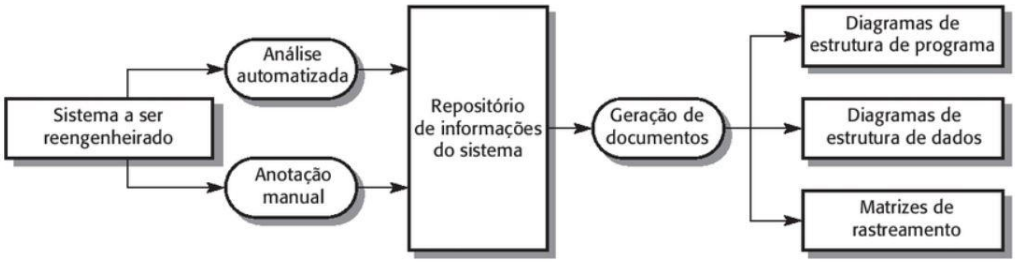
Waters e Chikofsky (1994) definem a engenharia reversa como aglomerado de tarefas dedicadas a entender e modificar um sistema, seja computadorizado ou não. No caso de softwares, as tarefas crucias são a identificação de componentes e suas relações e a descrição de alto nível de vários aspectos do sistema. Chikofsky e Cross (1990) complementam que nem sempre é necessário a existência de um sistema funcional para aplicar a engenharia reversa, é possível partir de qualquer nível de abstração de um sistema para obter mais informações. Ainda, engenharia reversa é um processo exclusivamente de examinação, não cabendo a ele modificação e nem implementação do sistema que passou por engenharia reversa.

Na figura 5 é mostrado de maneira básica o processo de uma engenharia reversa. Nela podemos observar que o sistema que irá passar por uma reengenharia pode ser analisado de maneira automatizada com uso de softwares de engenharia reversa ou feita de maneira manual. Todavia, as informações capturadas são lançadas para um repositório de

informações coletadas do sistema. Com os dados obtidos pode-se gerar alguns documentos para facilitar o entendimento do programa, como diagramas de estrutura, diagrama de dados e entre outros. Pressman (2016) complementa a explicação de processo de engenharia reversa mostrando três abordagens diferentes que podem ser empregadas:

- Engenharia reversa de dados: busca-se compreender a estrutura de dados interna e externas do programa. Na parte interna é feita a definição de classes de objetos e suas variáveis de programas. Os dados externos são objetos relacionados com banco de dados, sendo importante entender a estrutura quando deseja-se migrar para outro tipo de banco de dados ou mudança na estrutura.
- Engenharia reversa de processamento: nesta abordagem o objetivo é entender e extrair os procedimentos abstratos realizados pelo código fonte. Antes de realizar este processo é necessário um entendimento global de toda a aplicação para definir uma base de contexto. Dentro desse programa global, existem diversos sub-programas que executam processos abstratos que compõem a solução. É necessário criar um diagrama de blocos mostrando as interações destas abstrações funcionais.
- Engenharia reversa de interface de usuário: atividade mais comum no processo de reengenharia, dado que interface de usuário é um item quase que obrigatório para sistemas baseados em computadores. Algumas atividades são sugeridas, como buscar quais ações básicas a interface deve processar e suas devidas respostas.

Figura 5: Processo de engenharia reversa



Fonte: (SOMMERVILLE, 2004)

Comercialmente existem várias ferramentas automatizadas de engenharia reversa (MüLLER et al., 1995). O uso destas ferramentas apoiam o processo de entendimento do código, pois disponibilizam uma fácil navegação pelo código-fonte do software (SOMMERVILLE, 2004). A escolha de uma ferramenta de auxílio depende do tipo de informação que necessitamos obter em nosso sistema. Eilam (2005) em seu livro disponibiliza para conhecimento um grande acervo de software para realizar engenharia reversa, como por exemplo *IDA Pro* e *ILDasm*. Essas ferramentas no caso deste trabalho não ajudam tanto, pois o foco dos programas mencionados é desmontar o código fonte do programa e atuar dentro do código de máquina que é gerado. Já Pressman (2016) cita alguns softwares de engenharia reversa voltados a gerar representações estruturais e comportamentais a partir de código fonte, condizendo mais com o foco deste trabalho. Os softwares citados são: *Understand* desenvolvida pela *SciTools* e *Imagix 4D* desenvolvida pela *Imagix*.

2.3. Engenharia avante

Após realizado o processo de engenharia reversa e atingido o grau de abstração desejado, como mostrado na figura 3, é hora de iniciar a engenharia avante. Ela pode ser definida como o processo normal de desenvolvimento de um software, mudando apenas a camada de abstração que iremos partir neste desenvolvimento. Se na engenharia reversa, atingimos a camada de requisitos do sistema, partimos daqui a nova implementação do sistema. Em suma, todo conhecimento gerado anteriormente como abstrações e modelos serão implementados fisicamente. Aqui contudo deve-se atentar a evitar a inclusão de novas funcionalidades ou até mesmo mudança, pois pode complicar o processo de validação. Aplica-se nesta etapa todas as técnicas e disciplinas normais de desenvolvimento de software, como gerência de configuração, qualidade de software e testes (ROSENBERG, 1996), (CHIKOFSKY; CROSS, 1990).

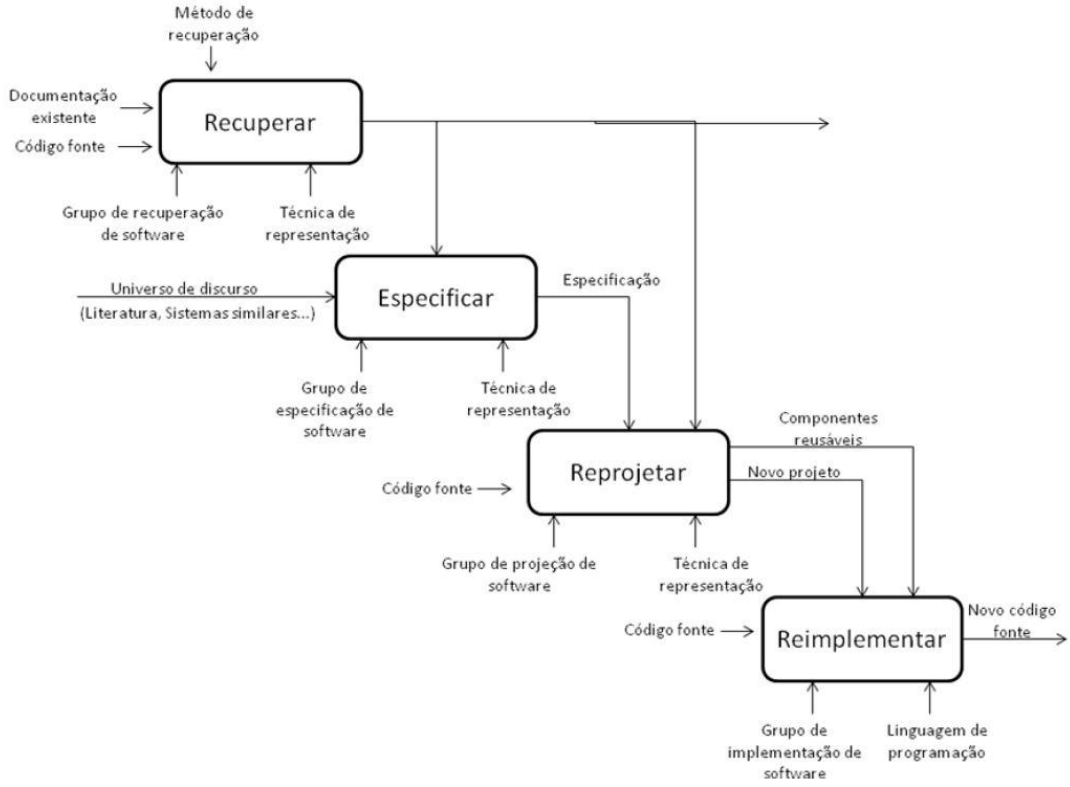
2.4. Trabalhos relacionados

Para ajudar no entendimento do principal tópico deste trabalho e buscar ideias no âmbito da reengenharia, foi feita uma pesquisa para buscar trabalhos relacionados com o tema do artigo. De maneira geral, não há muitos trabalhos com o tópico de reengenharia nos últimos 10 anos, apesar da importância do tema em um contexto de sociedade altamente dependente de software. Observa-se uma alta produção de conteúdo e estudos realizados nos início dos anos 90, podendo observar a data dos artigos dos autores referência no tema como Osborne e Chikofsky (1990), Arnold (1989) e Chikofsky e Cross (1990). Para esta seção buscou-se trabalhos mais recentes no tema, afim de usar alguma referência atualizada para o desenvolvimento do trabalho.

A monografia apresentada por Sousa e Leite (2012), mostra a aplicação de reengenharia para recuperar uma ferramenta de modelagem com código aberto chamada de *Oryx*. O objetivo dos autores era criar *plugins* para a ferramenta suportar novas linguagens de modelagem, uma vez que a ferramenta foi totalmente descontinuada pelos criadores originais. Para atingir o objetivo, os autores realizaram um processo de reengenharia na ferramenta, afim de recuperar o projeto original do software e reimplementa-lo com novos requisitos necessários para atingir os objetivos propostos. A sistemática de reengenharia utilizada pode ser vista na figura 6, foi proposta por Leite (1996, apud, SOUZA; LEITE, 2012, p.18) que segue em grande parte a teoria e procedimentos de reengenharia já apresentados neste trabalho. A parte principal do trabalho foi a aplicação de engenharia reversa, onde foi divida em três partes. A primeira consistiu em descobrir como instalar e preparar o ambiente para o software alvo. Na segunda foi feita busca por documentação de arquitetura do software e partir disso construir a base necessária para construir *plugins*. A terceira parte consistiu em estudar a codificação do ambiente de programação da ferramenta. Por fim os autores geraram um diagramas para mostrar as principais relações entre os componentes e os requisitos identificados. A parte de reimplementação é bem curta, pois segundo os autores, não seria necessário reimplementar toda a ferramenta, e sim apenas partes pertinentes a inclusão da nova funcionalidade de *plugins*. Grande parte do trabalho é dedicado a seção de testes dos *plugins*, onde os autores validaram a nova funcionalidade de *plugins* desenvolvida e evidenciar *bugs* pertinentes ao software original.

O trabalho desenvolvido por Almeida (2009) mostra uma aplicação prática de reengenharia, com o objetivo de implementar uma nova camada de persistência de ob-

Figura 6: Sistemática de reengenharia



Fonte: (SOUSA; LEITE, 2012)

jetos. O modelo de reengenharia adotado pelo autor é descrito no livro de Pressman (2006, apud, ALMEIDA, 2009, p.46) que segue os seguintes passos: análise de inventário, reestruturação de documentos, engenharia reversa, reestruturação de programas e dados e engenharia avante. O autor elaborou diagramas de classes e de uso com as informações obtidas na engenharia reversa. Montou também todo um mapeamento de diagrama de classes que serão persistidas. O trabalho de implementação realizado consistiu em reestruturar o código para implementar a nova camada de persistência. Contudo, o autor não apresentou nenhuma forma de validação da reestruturação realizada.

Um estudo real de reengenharia de software legado e com reimplementação total do sistema foi realizado por Bernhartr et al. (2012). Nele é descrito a reengenharia de um software de aeroporto com aproximadamente 40 anos de idade. O sistema em questão estava escrito em COBOL e não havia documentação e não possuía uma interface de uso amigável, sendo todo via linha de comando. Os autores utilizaram uma abordagem incremental para realizar a reengenharia com operação paralela de ambos software. No estudo foi descrito todo o processo de planejamento, análise e projeto de reimplementação do software. O processo foi exitoso, e um dos fatores que ajudaram foi o teste no ambiente de produção do software.

Cada um destes trabalhos utiliza a reengenharia para um fim diferente. O caso desenvolvido por Sousa e Leite (2012), usou para entender o sistema legado e desenvolver uma nova funcionalidade para este sistema. Já Almeida (2009) visou entender o sistema para propor uma nova camada de persistência. Em ambos trabalhos, a metodologia de

pesquisa empregada foi uma pesquisa-ação. Entretanto, não houve uma reimplementação completa de sistema. Já o caso apresentando por Bernhartr et al. (2012) é o que mais se aproxima do trabalho que será demonstrado aqui. O caso de Bernhartr et al. (2012) é uma aplicação mais clássica de reengenharia, onde temos um sistema legado em uso muito antigo, escrito em outro paradigma de programação e com tecnologias que já ultrapassadas. É considerado também uma pesquisa-ação com a aplicação prática de reengenharia completa em um sistema. A abordagem escolhida é a de modo incremental, onde o sistema foi quebrado em estruturas que passavam por reengenharia e entregues ao cliente para testes daquela estrutura em específico.

No trabalho aqui proposto, trata-se de um sistema mais novo e desenvolvido em tecnologia atual. O software será refeito visando corrigir os erros de implementação e torna-lo mais focado no seu negócio. A abordagem escolhida para execução da tarefa se assemelha a definida por Rosenberg (1996), chamada de *Big Bang*, pois será feita uma *release* com todas as funcionalidades para o cliente.

3. Metodologia da Pesquisa

A natureza multidisciplinar da engenharia de software torna a escolha de uma metodologia pesquisa uma tarefa importante, pois é através do método escolhido que define-se como pesquisar e apresentar os resultados. Um trabalho de engenharia de software pode envolver o estudo do indivíduo que usa o software, estudo de equipes que desenvolvem um software e as tecnologias de desenvolvimento. Portanto temos diversas áreas, passando pela psicologia e até sociologia (EASTERBROOK et al., 2007).

3.1. Delineamento de pesquisa

O enfoque desta pesquisa foi qualitativo, pois visou abordar o mundo "lá fora" e não em contextos especializados de pesquisa, como em laboratórios. Busca-se nesse método analisar a experiência de indivíduos ou grupos, examinar as interações e comunicações desenvolvidas e investigar documentos, sejam textos, imagens, filmes ou músicas (FLICK, 2009). A pesquisa qualitativa encaixa-se no contexto deste trabalho, pois aqui foi visado a aplicação da pesquisa em um ambiente estritamente particular, ou seja, que possui suas próprias características e grupo social (RICHARDSON, 1999 apud AZEVEDO et al., 2011).

Baseado na questão que este trabalho pretende responder, o nível da pesquisa é definido como exploratório. Pesquisas deste nível são aplicáveis quando buscamos conhecer, levantar ou descobrir informações sobre uma área ou tema de trabalho (AZEVEDO et al., 2011).

O método de pesquisa utilizado foi a pesquisa-ação. De acordo com Easterbrook et al. (2007) o uso da pesquisa-ação é ideal em cenários onde o pesquisador não apenas observa o fenômeno em estudo, mas também intervém com o objetivo de melhorar a situação. A presença do dono do problema é fundamental neste método, para que ajude na identificação e solução do problema.

3.2. Unidade de análise

A pesquisa ocorreu dentro da empresa Comlink Equipamentos Eletrônicos, localizada em Caxias do Sul. O sistema alvo desta pesquisa trata-se de um software para *desktop*

chamado de Gravador de Produtos, que tem por finalidade facilitar a gravação de microcontroladores embarcados nos produtos eletrônicos fabricados pela Comlink. O software é implementado no setor de produção com o seu uso ocorrendo na fase final de montagem de um produto.

A pesquisa contou com o apoio de colaboradores da empresa para validar o sistema reimplementado. No setor de produção foram destacados dois colaboradores, sendo um o gerente de produção para realizar o cadastramento de produtos no sistema e um operador de produção para realizar a gravação de produtos. Teve ainda apoio de um colaborador do setor de pesquisa e desenvolvimento que ficou como responsável pela instalação e acompanhamento do uso do software pelo setor de produção. No total foram envolvidos diretamente pela pesquisa três funcionários da empresa.

3.3. Coleta de dados

Para coletar os dados de validação do software foi utilizado o método de entrevistas feitas de maneira semi-estruturada. Desta maneira a entrevista segue um fluxo de conversa, onde novas questões surgem conforme a entrevista avança. Geralmente, a entrevista semi estruturada não segue questões, e sim tópicos. O método escolhido tem bastante utilidade para avaliar e levantar requisitos de um software (SINGER et al., 2008).

As entrevistas iniciaram no dia 02/05/19 com a liberação da primeira *release* e prosseguiram até 10/06/2019 após a liberação da terceira *release*. Elas foram realizadas semanalmente ou quando surgia alguma dificuldade no uso do software. As entrevistas foram feitas com o colaborador do setor de pesquisa e desenvolvimento da empresa e seguiu os seguintes tópicos:

- Resumo do uso do software durante a semana;
- Se aconteceram problemas, solicitar o *log* de passos executados pelo software, produtos em gravação e gravador utilizado;
- Necessita de alguma melhoria na interface ou ajuste em alguma funcionalidade?

As respostas foram obtidas através de *e-mails* por ser mais fácil de enviar os *logs* de informação que o software gera durante sua operação.

3.4. Análise de dados

Após a realização das entrevistas, os dados coletados devem ser analisados para guiar os próximos passos do sistema. Para isso foi empregado a técnica de análise de conteúdo, com o objetivo de entender e buscar as melhorias necessárias ao sistema reimplementado.

A análise de conteúdo é um conjunto de técnicas utilizadas para verificar dados qualitativos presentes em textos de entrevistas, reportagens e questionários. Tem por objetivo descrever o conteúdo das mensagens contidas no documento, buscando o significado oculto presente no conteúdo (BARDIN, 1995 apud AZEVEDO et al., 2011).

4. Desenvolvimento

Este capítulo mostra a aplicação do processo de reengenharia no software escolhido. Inicialmente é realizado um estudo geral sobre o software alvo e após é realizado a engenharia reversa visando recuperar os requisitos originais do sistema. Com estes novos requisitos o sistema é reimplementado em uma nova forma.

4.1. O Gravador de Produtos

O software escolhido para passar por um processo de reengenharia é denominado de Gravador de Produtos. O sistema em questão está instalado numa pequena empresa de manufatura de produtos eletrônicos, situada em Caxias do Sul, no Rio Grande do Sul. A empresa, que atua a 20 anos no mercado, possui um portfólio de produtos bastante variado, desde produtos para o ramo de implementos rodoviários até os residenciais.

A grande maioria destes produtos eletrônicos possuem microcontroladores, que por sua vez possue um software embarcado, também conhecido por *firmware*. Após realizada a montagem dos componentes eletrônicos, é necessário gravar o *firmware* no microcontrolador do equipamento. É nesta etapa de produção que entra o software alvo deste trabalho, o Gravador de Produtos.

Este software funciona como uma interface de gravação para os operadores de produção da empresa. Através do sistema, os operadores selecionam um código de produto, conforme a ordem de produção da peça, verificam qual hardware de gravação é necessário utilizar e executam a gravação do *firmware* através do software. O operador ainda pode verificar se a operação terminou com sucesso ou algum problema ocorreu.

O motivo do uso deste software é facilitar o trabalho dos operadores. Dentro da empresa Comlink, atualmente está em uso três fabricantes de microcontroladores, que são *Microchip*, *STMicroelectronics* e *Cypress*. Cada um desses fabricantes possui um software gravador próprio, que realiza a comunicação com o hardware de gravação. Para uma pessoa sem muita experiência na área, a escolha do software correto pode causar uma certa confusão.

Estes softwares são designados para uso de pessoal de desenvolvimento de produtos, pois além de gravar estes programas podem fazer verificação de memória, apagar setores de memória do microcontrolador e entre outras várias funções. O que um operador de produção deseja é apenas gravar o seu produto, por isso funções além do desejado mais atrapalha do que ajuda. Com a função desejada selecionada o operador deve ainda selecionar o *firmware* para ser gravado dentro de pastas do diretório de rede da empresa. Portanto, através do Gravador de Produtos o operador consegue executar todos os passos citados de uma maneira simplificada e única, não tendo nenhuma divergência por causa de diferentes tipos de fabricantes de microcontrolador.

4.1.1 Hardwares de gravação

Como já citado anteriormente, a Comlink trabalha com três tipos de fabricantes diferentes de microcontroladores. Cada um destes fabricantes disponibiliza uma forma de gravar, depurar e acessar memória de seus microcontroladores. Fabricantes como *Cypress* e *STMicroelectronics* utilizam em seus microcontroladores a arquitetura de processador *ARM (Advanced RISC Machine)*. Para acessar a memória destes microcontroladores em específico, é necessário realizar uma conexão física até uma porta de *debug*. Esta porta suporta protocolos de comunicação do tipo *JTAG (Joint Test Action Group)* ou *SWD (Serial Wire Debug)* (ARM, 2017). Através desta conexão é possível realizar operações na memória, como leitura, escrita e depuração de código gravado.

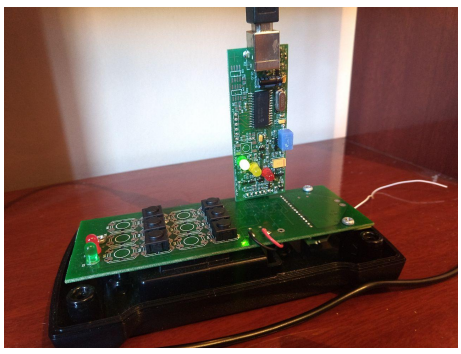
A Comlink também utiliza microcontroladores da fabricante *Microchip*, que por

sua vez possuem uma arquitetura própria. Isto acarreta em mudanças na forma de manipular memória de um microcontrolador, sendo necessário outro tipo de protocolo, ferramenta de gravação e software de controle da ferramenta. O que é comum entre estes três fabricantes é a necessidade de existir um software que acesse o gravador físico e transmita os comandos de operação de memória para o microcontrolador. Cada fabricante tem um software para acessar seus gravadores, onde o operador tem acesso a uma gama de funções para operar na memória do microcontrolador. Os fabricantes fornecem também uma versão de *Command Line Interface* (CLI) de cada programa gravador. Estes programas recebem através de parâmetros os comandos para executar através da interface de linha de comando do sistema operacional. Para exemplificar, foi pego uma amostra do *set* de comandos do manual de usuário da STMicroelectronics (2018) e montada a lista abaixo.

- Comando: -ME
  - Descrição: Executa apagar geral de memória do *chip*.
  - Sintaxe: `-ME`.
- Comando: -SE
  - Descrição: Apaga setores da memória *flash*.
  - Sintaxe: `-SE <inicio_setor> [<fim_setor>]`.
- Comando: -P
  - Descrição: Carrega um arquivo do tipo binário, Inte Hex ou Motorola S na memoria de um microcontrolador sem verificar.
  - Sintaxe: `-P <caminho_arquivo> [<endereco>]`.

O software Gravador de Produtos acessa o gravador físico dos fabricantes através dos programas do tipo CLI. A figura 7 mostra um gravador acoplado em um produto da *Comlink*, o gravador em questão é chamado de *Pickit* e é designado para gravar produtos da *Microchip*.

Figura 7: Gravador *Pickit*



Fonte: do autor

4.1.2 Motivos para Reengenharia

O software Gravador de Produtos tem aproximadamente 10 anos de idade. Surgiu de uma solicitação do departamento de produção que gostaria de unificar a interface de acesso aos gravadores de produção. Esta solicitação foi realizada informalmente, através de conversa

entre o solicitante e os desenvolvedores. Por tanto, não existe documentação referente a concepção do projeto e os requisitos de usuário solicitados.

O Gravador de produtos é um programa do tipo *Windows Forms*, desenvolvido na linguagem C♯ através do ambiente de desenvolvimento *Visual Studio* da *Microsoft*. No que tange a ferramentas e paradigma, o projeto foi feito com tecnologias atuais, portanto não há nenhuma objeção quanto a isso. O problema reside na maneira que o programa foi desenvolvido. Há funcionalidades que não são utilizadas, dados de programa que apenas confundem os usuários, problema de compatibilidade se instalado em outras máquinas e questões importantes como localização de banco de dados e acesso restrito estão embutidos no código do programa. Se for necessário trocar o local do banco de dados ou mudar a senha de acesso privilegiado, é necessário recompilar o projeto.

Ultimamente a empresa vem buscando mudar alguns de seus processos e algumas destas mudanças envolveriam modificações no software. Tais mudanças podem não ser fáceis de executar devido a organização do código. Boa parte lógica de funcionamento está codificada na interface do programa, como irá mostrar o processo de engenharia reversa. Com isso, o risco de modificar algo e estragar alguma funcionalidade não correlata é alto.

Outro ponto é o própria ferramenta de desenvolvimento, o *Visual Studio* que não é mais utilizado pela empresa, tendo o seu foco mudado para outras plataformas de desenvolvimento. A empresa não renovou sua licença profissional e a versão gratuita disponibilizada pela *Microsoft*, não pode ser utilizada pela Comlink, pois ela não se encaixa nos requisitos solicitados pela *Microsoft*.

4.2. Metodologia

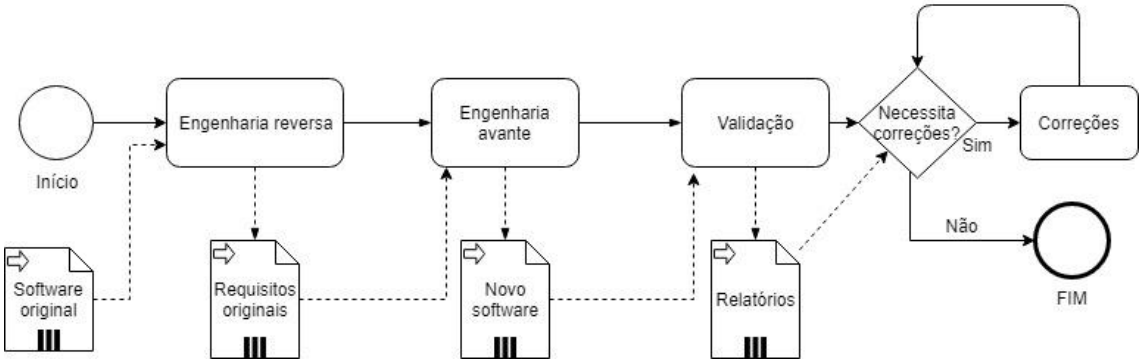
O processo de trabalho definido foi baseado nos estudos realizados dentro da seção de fundamentação teórica. O trabalho pode-se dividir em três grandes partes, sendo:

- Engenharia reversa: necessário para abstrair as camadas do software. Será iniciado a partir da camada de implementação, onde será analisado as interfaces do programa, código fonte e dados, para então chegar nos requisitos originais. Nesta camada é feita uma limpa nos requisitos para reimplementar aquilo que é útil e funcional para o projeto. Serão feitos diagramas dos dados recuperados nas análises;
- Engenharia avante: reimplementação dos requisitos levantados e modificados na etapa anterior. Os componentes reimplementados serão mostrados através de diagramas para ficar clara suas relações. Haverá também uma descrição das tecnologias utilizadas para reimplementação;
- Validação: para validar o software antes de entregar a empresa, serão testados os requisitos implementados. Após, o software será testado pelo departamento de engenharia da Comlink, aonde será avaliada a nova interface e funcionalidades. Se problemas forem encontrados, estes serão prontamente resolvidos e indicados no trabalho. Após testes de engenharia, será a vez do setor de produção testar em seu ambiente de trabalho. Novamente, se problemas surgirem serão devidamente solucionados.

As etapas descritas acima podem ser visualizadas no diagrama 8. Cada processo vai gerar um documento que será utilizado no processo seguinte. A finalização ocorre

quando não houver mais demandas por alterações para entrar em conformidade com software atual.

Figura 8: Diagrama de processos



Fonte: do autor

4.3. Aplicação da engenharia reversa

A engenharia reversa tem por objetivo buscar o entendimento do software e gerar base de conhecimento para reimplementação. O procedimento seguido terá como referência as abordagens mostradas por Pressman (2016). Será feito em três fases, que são:

- Dados do programa: analisar os dados que são salvos no banco de dados, buscando cada atributo e sua aplicação dentro do programa. Verificar também dados internos que são utilizados nos processos;
- Processo: levantar os principais componentes do programa e como eles se relacionam internamente. Buscar um entendimento da arquitetura geral do software;
- Interface: analisar as interfaces de usuários e as ações realizadas por cada elemento;

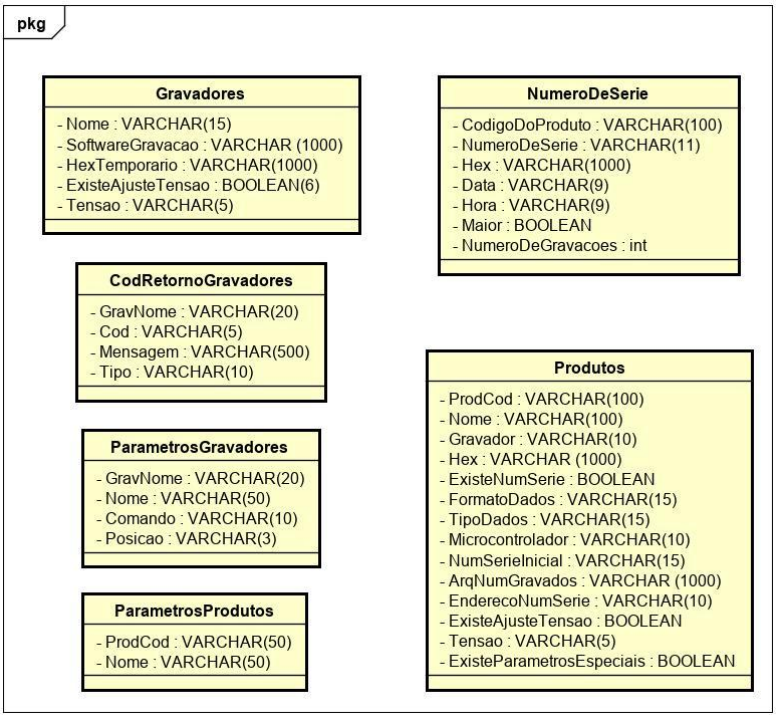
4.3.1 Engenharia reversa de dados

Como já foi mencionado, busca-se aqui entender e documentar na engenharia reversa de dados toda a estrutura de dados do programa. Começando pela estrutura externa de dados, aquelas que são salvas em bancos de dados. Para este tipo de armazenamento, o software utiliza um banco de dados do tipo *SQLite*, que é parecido em muitos aspectos com bancos do tipo *MySQL*. A diferença é que *SQLite* é ideal para casos que o tráfego de requisições ao banco é baixo, ou seja, poucas aplicações acessando o banco, pois o foco *SQLite* é ser um banco leve para embarcar em dispositivos e por isso ela dispensa camadas de controle de acesso ao banco.

A localização deste banco de dados fica em uma pasta de rede da empresa. O acesso é restrito a membros com permissão. O motivo do arquivo de banco de dados estar na rede é para garantir que seja feito o *backup* rotineiramente. O software gravador de produtos acessa o banco de dados através de requisições do tipo *SQL* e o caminho de acesso fica embutido no código fonte. Caso o banco mude nome ou troque de lugar, é necessário alterar o código fonte do software e compilar.

Foi realizada uma cópia do arquivo de banco de dados da empresa para realizar a engenharia reversa. Para facilitar foi utilizado uma ferramenta chamada de *DB Browser for SQLite* que permite visualizar as tabelas e os dados salvos dentro do banco. As tabelas presentes dentro do banco de dados são objetos de classes do programa, portanto será representado como uma classe. A figura 9 representa as tabelas neste formato.

Figura 9: Classe de dados



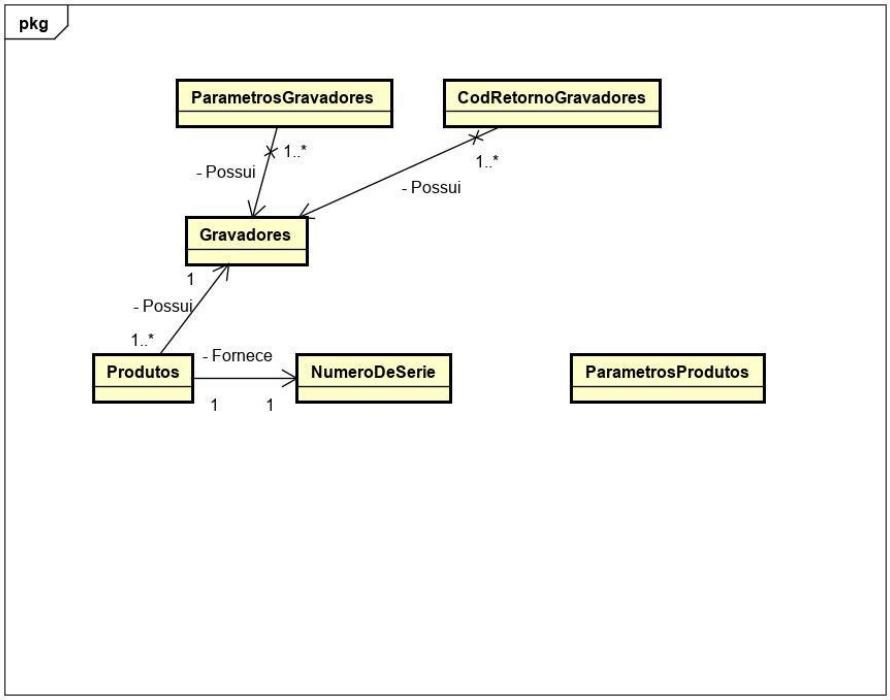
Fonte: do autor

De maneira sucinta, a funcionalidade de cada classe de dado é descrita da seguinte forma:

- Gravadores: possui todos atributos de um gravador armazenado. Chave primária: Nome;
- CodRetornoGravadores: possui todos atributos de retorno de um gravador. Tem como chave estrangeira proveniente de Gravadores;
- ParametrosGravadores: possui todos atributos de parâmetros de um gravador. Tem como chave estrangeira proveniente de Gravadores;
- Tabela ParametrosProdutos: sem uso;
- NumeroDeSerie: armazena data, hora, código de produto e número série de um produto gravado. Necessita de parâmetro de gravadores e outro inserido pelo usuário;
- Produtos: possui todos atributos de um produto armazenado. Possui chave primária Nome e estrangeira que o Gravador relacionado com o produto

A relação entre as classes de dados externos é mostrado na figura 10. Um Produto vai sempre possuir um Gravador atrelado. CodRetornoGravadores e ParametrosGravadores possuem um Gravador associado. O Gravador por sua vez não conhece CodRetornoGravadores e ParametrosGravadores diretamente. NumeroDeSerie recebe de Produtos o atributo ProdCod para atualizar sua tabela, indicando que um novo produto foi gravado.

Figura 10: Relação entre classe de dados externos



Fonte: do autor

4.3.2 Engenharia reversa de processamento

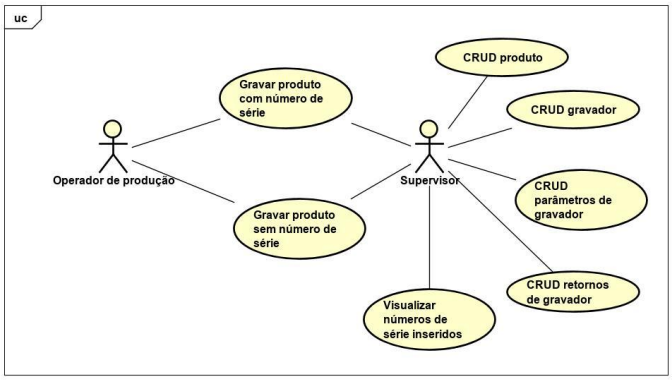
O objetivo da engenharia reversa de processamento é entender os processos realizados pelo código fonte, ou seja, busca-se identificar os componentes que existem e descrever seu funcionamento. Mas antes, é necessário ter uma visão global de funcionamento do programa, ficando mais fácil o entendimento e evidente os componentes que existem dentro dele (PRESSMAN, 2016).

O usuário ao executar o programa tem acesso a tela principal do sistema, onde ele pode gravar um produto, inserir número de série no produto se necessário e acessar menus para cadastrar gravadores, produtos, parâmetros e retornos de gravador. Atualmente há dois tipos de usuários no sistema, sendo um considerado operador de produção que não possui privilégio para realizar cadastros ou consultas e o supervisor que realiza a parte de consultas e cadastros. A figura 11 mostra o diagrama de caso de uso do software.

O software foi desenvolvido utilizando o programa *Visual Studio* e foi programado na linguagem `C#` dentro de um paradigma orientado a objetos. A organização das classes dentro do projeto foi feito conforme mostra na figura 12. Cada pasta tem a seguinte função:

- BLL: armazena a classe BussinessLayer, servindo como isolamento da camada de banco de dados;
- Dao: o nome sugere o uso do padrão *Data Access Object*, entretanto não é bem identificável o uso deste padrão no projeto. A classe armazenada possui métodos de acesso ao banco de dados;
- Gui: armazena as classes de interface com o usuário, chamadas de *Windows*

Figura 11: Casos de uso do software

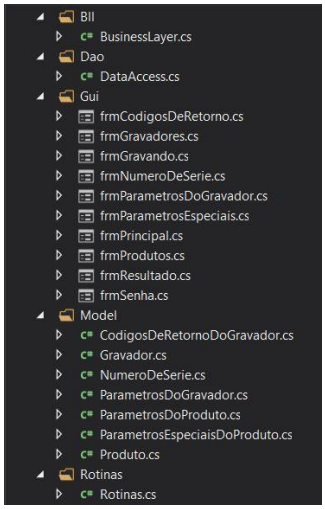


Fonte: do autor

Forms;

- Model: armazena todos os modelos de dados vistos no capítulo de engenharia reversa de dados;
- Rotinas: armazena classe de rotinas comuns que são utilizadas pelas outras classes do programa;

Figura 12: Estrutura do projeto



Fonte: do autor

Quase toda a lógica de funcionamento está dentro das classes da pasta Gui, desta forma fica difícil montar um diagrama de componentes com as classes. Cada arquivo *form* contém métodos de lógica de negócio que são executados em ações do usuário, como no *callback* executado ao clicar em um botão da interface.

O *frmPrincipal.cs* contém o principal processo do software, que é a gravação de um produto. O apêndice A mostra o diagrama de atividades elaborado através da análise da execução deste processo e abrange os principais atores que são o operador, software e o banco de dados. Nele foi possível perceber a atuação dos principais dados do sistema e também a ausência de uso da classe de dado *CodigosDeRetornoDoGravador.cs*

Os *forms* descritos abaixo executam basicamente a mesma função, que é de cadastrar, deletar, editar e alterar (CRUD) a classe de dados externos à eles atrelado. São eles:

- *frmCodigosDeRetorno*: operações de CRUD em CodigosDeRetornoDoGravador;
- *frmGravadores*: operações de CRUD em Gravador;
- *frmNumeroDeSerie*: apresenta os dados presentes em NumeroDeSerie;
- *frmParametrsoDoGravador*: operações de CRUD em ParametrosDoGravador;
- *frmProdutos*: operações de CRUD em Produto;

Fora das operações de CRUD, temos o *form frmSenha* que faz a interface para inserção de senha para acessar todos as telas que realizam CRUD nas classes de dados externos. Ele realiza também a verificação de senha, que está embutida de forma rígida no código fonte da classe.

Fora das classes *forms* existem outras que operam uma pequena parte do programa. Dentro das estruturas BLL e Dao estão as classes *BussinessLayer.cs* e *DataAccess.cs* que realizam as operações do banco de dados. Tanto a classe *BusinessLayer.cs* quanto *DataAccess.cs* foram projetadas no padrão de *singleton*, ou seja, só é possível criar uma instância delas no programa. A classe *DataAccess.cs* tem a implementação do tipo de banco de dados utilizado e a localização embutida de forma rígida no código fonte. Se for algum dia necessário trocar a localização, o código do software deve ser alterado e recompilado. Possui também todos os métodos que realizam *Query* no banco. Fica evidente a inteção do projetista em ter feito um modelo de padrão *Direct Access Object* para o banco, mas por motivos desconhecidos não foi feito da maneira correta.

Já a classe *BusinessLayer.cs* possui os métodos que são acessados pelos *forms* para buscar dados do banco. Estes métodos por sua vez acessam aqueles métodos da classe *DataAccess.cs*, portanto a classe *BusinessLayer.cs* serve como uma camada de isolação do acesso ao banco de dados.

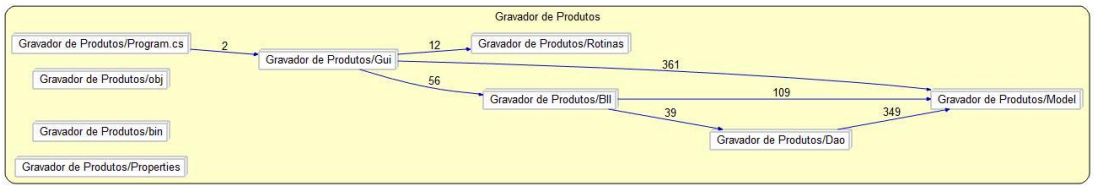
A última classe auxiliar é chamada de *Rotinas.cs* que possui apenas um método para verificar o *checksum* de um arquivo hexadecimal.

Para entender a relação de dependência entre as classes, foi utilizado uma ferramenta de análise de código chamada de *Understand* da empresa *SciTools*. O motivo do uso foi para complementar a análise de como as estruturas do projeto estão relacionadas entre si. A figura 13 mostra a como está a relação de classes das estruturas. A imagem confirma algumas constatações da análise como o uso das classes do BLL para acesso exclusivo ao Dao e a centralização do uso de classes do Model. O programa fornece outras informações como contagem de linhas de código que neste projeto ficou em 5041 linhas, diagramas UML e entre outros.

Após toda a análise foi possível identificar todos os processos executados pelo programa e a arquitetura em que foi projetado. Os processos identificados foram:

- Manipulação de Hex: processo que é invocado quando é necessário inserir número de série no produto;
- Chamada de processos externos: processo executado quando o programa de controle do hardware é chamado. Os parâmetros cadastrados para este gravados são passados neste processo. O sucesso da gravação ocorre no retorno do programa executado;

Figura 13: Relacionamento entre estruturas



Fonte: do autor

- Acesso ao banco de dados: processo que abre e realiza as consultas necessárias ao banco de dados;
- *CRUD* diversos: realizar operações de *CRUD* nos dados externos do programa;

4.3.3 Engenharia reversa de interface

Esta etapa consiste em analisar as interfaces de usuário implementadas no software legado. Como já comentando a respeito das estruturas do projeto, todas as interfaces estão implementadas dentro da pasta de Gui. Cada um dos *forms* representa uma tela de interação com o usuário.

A primeira tela para análise é mostrada na figura 14, ela representa a principal tela do sistema e está dentro da classe *frmPrincipal*. Ela possui uma aba superior com três opções, sendo uma contendo as telas para cadastros dos dados do programa. Logo abaixo está o campo de seleção de produto, onde o operador procura um produto através do código de cadastro. Após a escolha do produto, dentro do campo de detalhes aparece informações referentes ao produto. A informação mais útil para o operador é de qual gravador ele deve utilizar. Embaixo do campo de informações está a parte de inserção de um número de série, que só é habilitado quando o produto possui número de série. Possui dois campos, sendo um chamado de número de série anterior e o outro de número a ser gravado, o usuário insere neste último. Existem três botões ao lado para realizar operações nos números de série, sendo de incrementar, decrementar e editar. Este último destrava e trava os campos para edição. Originalmente era para inserir números no formato hexadecimal, mas foi mudado para decimal em um campo que foi improvisado logo ao lado do texto de ”Número de série a ser gravado”. Por fim, na base existe o botão de ”Gravar”que iniciar todo processo de gravação. Caso algum erro ocorra, como por exemplo um caminho para arquivo hexadecimal não existente, uma mensagem de erro com a descrição do problema é lançada ao usuário. Ligado a operação de gravação existe uma tela que informa através de uma barra de progresso o estado da gravação, ao final esta mesma tela informa se o último estado foi sucesso ou erro.

Já as telas que envolvem operações do tipo *CRUD* em objetos de dados externos seguem um mesmo padrão de interface conforme mostra figura 15. Possuem uma lista com os itens cadastrados para selecionar, caixas de texto para inserir novos valores aos atributos e cinco botões com as seguintes ações:

- Salvar: salva o item criado ou alterado;
- Novo: inicia a criação de um novo item;
- Apagar: apaga o item selecionado;

Figura 14: Tela principal

Fonte: do autor

- Limpar: limpa todos os campos para inserir atributos ao item;
- Alterar: inicia atualização do item selecionado;
- Sair: voltar para tela principal;

Da mesma forma que a tela principal, caso ocorra algum erro uma janela auxiliar informa o problema ocorrido.

A tela de informação dos números de série gravados mostrada na figura 16 está codificada em *frmNumeroDeSerie* e tem o propósito de apenas mostrar dados sem realizar operações de *CRUD*. De um lado está disposto uma lista de produtos que possuem número de série e do outro os atributos atrelados a aquele produto.

A última tela é para inserir senha necessária para acessar as telas de cadastro e visualização dos dados do programa. É uma tela bastante simples onde existe apenas um campo para inserção de senha e um botão para acessar o módulo desejado. Caso a senha esteja errada, um texto em cor vermelha aparece informando erro na operação.

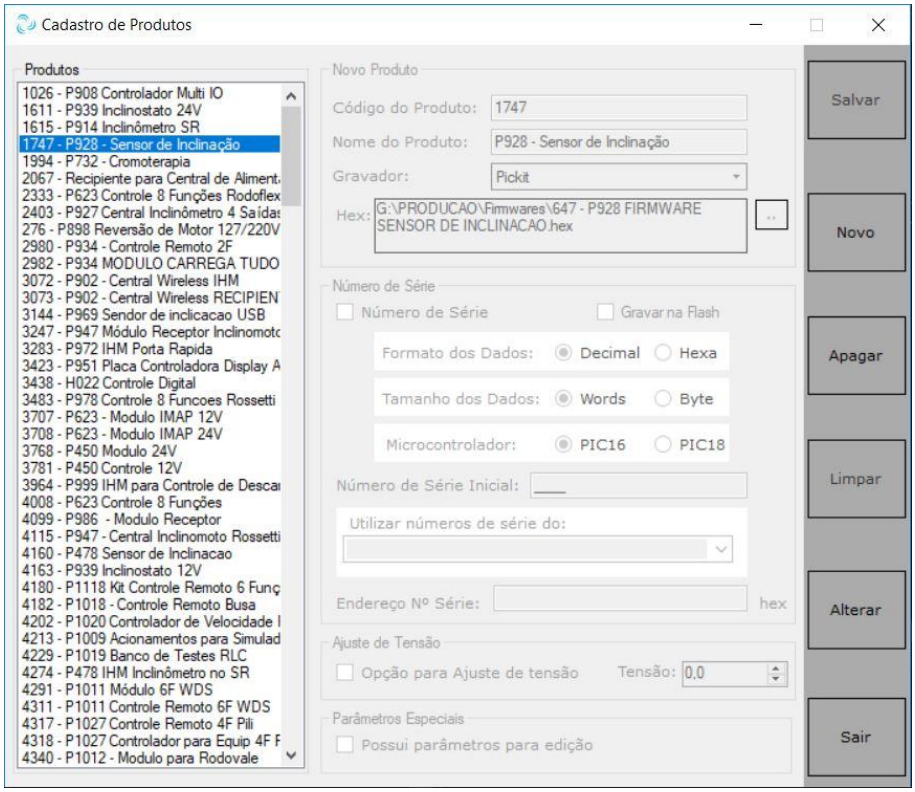
4.4. Requisitos abstraídos

Com o trabalho da engenharia reversa foi possível abstrair os requisitos implementados no software legado. Há dois tipos de requisitos: os funcionais, que são aqueles que agregam valor ao negócio e os não funcionais, que são aqueles que descrevem como o sistema irá executar os requisitos funcionais.

4.4.1 Requisitos funcionais

Os requisitos funcionais detectados no processo de engenharia reversa foram os seguintes:

Figura 15: Tela de produtos

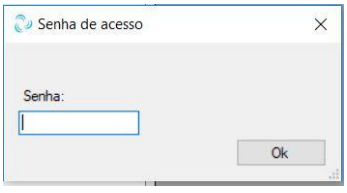


Fonte: do autor

Figura 16: Tela de números de série

Fonte: do autor

Figura 17: Tela de senha



Fonte: do autor

RF001 – CRUD Produtos

O sistema deverá possibilitar para usuários autorizados a cadastrar, atualizar e apagar os produtos para a serem gravados, com os seguintes dados:

- Nome do produto
- Código
- Local do arquivo hexadecimal
- Gravador utilizado
- Arquitetura do microcontrolador
- Se possui número de série
- Endereço do número de série caso possua número de série
- Formato e tipo dos dados de número de série que serão modificados no arquivo hexadecimal

RF002 – CRUD Gravadores

O sistema deverá possibilitar para usuários autorizados a cadastrar, atualizar e apagar os gravadores que são utilizados para gravar produtos. O gravador terá os seguintes atributos:

- Nome do gravador
- Localização do programa específico do gravador
- Parâmetros para usar no programa gravador
- Código de retornos do gravador
- Local para *hex* temporário

Os dados com estes atributos devem ser persistidos em um banco de dados.

RF003 – CRUD Parâmetros de gravador

O sistema deverá possibilitar para usuários a cadastrar, atualizar e apagar os parâmetros que são utilizados para executar os programas de gravação. Os seguintes atributos são definidos:

- Nome do parâmetro
- Parâmetro, que pode ser uma *string* ou um *char*
- Gravador relacionado
- Posição do comando no momento de montar a chamada do programa

Em parâmetros que o programa deve completar com o caminho do arquivo a ser gravador, será utilizado uma *string* especial, definida como "%H", assim o programa irá substituir esta *string* pelo caminho apontado pelo produto. Os dados com estes atributos devem ser persistidos em um banco de dados.

RF004 – CRUD Retornos de gravador

O sistema deverá possibilitar para usuários autorizados a cadastrar, atualizar e apagar retornos que são recebidos após a execução de um programa de gravação. Os seguintes atributos são definidos:

- Mensagem do retorno
- Código numérico do retorno
- Tipo de retorno, se foi ERRO ou SUCESSO
- Gravador relacionado

Os dados com estes atributos devem ser persistidos em um banco de dados.

RF005 – Listar números de série por produto
Para cada produto com número de série, deve-se disponibilizar uma interface para visualização dos números de série gravados para aquele produto. Os seguintes atributos são esperados: • Código do produto • Número de série gravado • Data e hora • Local do arquivo hexadecimal

RF006 – Inserir número de série nos arquivos hexadecimal
O sistema internamente deverá inserir o número de série inserido pelo usuário no endereço especificado pelo produto. Contudo, o arquivo original deverá permanecer intacto, portanto deve-se criar um arquivo temporário para inserir o número.

RF007 – Realizar interface com programas de gravação
O programa deve conseguir executar um programa do tipo *command line interface* (CLI) e serializar os parâmetros cadastrados do gravador.

RF008 – Exibir para o operador resultado da operação de gravação
O programa deve apresentar o resultado da operação de gravação para o operador. Em caso de erro, indicar de alguma forma o tipo de erro.

RF009 - Restringir acesso a telas de cadastro e visualização
O programa deve restringir o acesso as telas para cadastro e visualização de dados do sistema. A restrição deve ser feita através de senha de acesso.

RF010 - Pesquisar produtos pelo código
O operador deve conseguir pesquisar um produto para gravação através do código de produto.

RF011 - Disponibilizar informações do produto
Ao pesquisar e selecionar um produto algumas informações básicas devem ser disponibilizadas para o operador. As informações são: • Nome do produto • Gravador do produto • Localização do arquivo hexadecimal

RF012 - Inserção do número de série tanto em hexadecimal quanto em decimal
O operador deve conseguir inserir o número de série tanto em decimal quanto em hexadecimal.

RF013 - Incremento automático do número de série
Ao gravar um produto com número de série, o programa deve incrementar automaticamente o número de série inserido.

4.4.2 Requisitos não funcionais

Já os requisitos não funcionais detectados na engenharia reversa foram os seguintes:

RNF001 - Utilizar banco de dados *SQLite*
O banco de dados será do tipo *SQLite*, com isso as operações serão através de *Querys* de *SQL*.

RNF002 - Instalar banco em pastas de rede
A instalação do banco de dados deve ficar em pastas da rede interna da empresa para ser incluída no processo de *backup*.

RNF003 - Operar em sistemas Windows de 32 bits
Deve ser possível instalar e usar o software em sistemas operacionais Windows em 32 bits.

RNF004 - Otimizar programa para máquinas de baixo desempenho
No projeto, deve-se considerar desenvolver o software de maneira otimizada para equipamentos de baixo desempenho.

4.5. Reimplementação

O último processo da reengenharia visa reimplementar o software recuperado nas etapas de engenharia reversa em uma nova forma. Seguindo a ideia apresentada por Rosenberg (1996), ilustrada na figura 3, é necessário repensar os conceitos, reespecificar os requisitos, reprojetar e recodificar o sistema. A primeira parte vai trabalhar em cima dos requisitos abstraídos na engenharia reversa. Alguns dos requisitos levantados vão sofrer modificações para atender melhor o objetivo do sistema. O processo de reprojetar vai trabalhar em como iremos reimplementar o sistema com base nos requisitos abstraídos e repensados. A recodificação vai concentrar o esforço para concretizar todo o reprojeto levantado anteriormente.

4.5.1 Reespecificar

Dos requisitos levantados, existem alguns que necessitam passar por uma reespecificação e outros ainda necessitariam ser deixados de lado para atender melhor as expectativas da empresa e focar na área de solução do sistema, que é gravar um produto. Abaixo estão os requisitos que serão alterados:

- RF002: o atributo de localização do programa de gravação necessita sair do banco de dados. Este caminho será gravado em um arquivo de texto. Assim em cada computador que for instalado o software terá o seu caminho local para buscar o software de gravação.

- RF003 e RF004: no software atual estes dados são gravados em banco de dados. Para a reimplementação estes dados serão retirados do banco e colocados em arquivo de texto. O gravador irá ter como atributo o caminho dos arquivos de parâmetros e de retorno. No apêndice B é apresentado um modelo do arquivo de parâmetros e outro de retornos que um gravador deve possuir.
- RF009: hoje a restrição é através de uma senha gravada no código fonte. Necessário modificar este esquema para um serviço de autenticação completo com usuário e senha, restringindo os acessos através da permissão deste usuário. Com isso é necessário criar uma nova classe de dado para usuários. O atributo de senha ao ser salvo no banco de dados terá seu valor convertido através de funções de criptografia *hash* no modelo de algoritmo SHA-1, evitando assim a exposição explícita deste atributo.
- RNF002: hoje o endereço do banco de dados está rígido no código fonte do programa. É necessário alterar o caminho sem modificar o código fonte. Para isso, durante a inicialização o programa irá ler um arquivo de texto que contém o caminho do banco de dados. Este arquivo ficará armazenado na pasta de instalação do software.

Devido a necessidade da modificação nos requisitos RF003 e RF004, surge um novo procedimento ao utilizar o software, que é realizar um *login* de usuário antes de abrir a tela principal. O fluxograma de processo apresentado no apêndice A terá novas ações de execução, que são destacados em azul no apêndice C.

Diante a possibilidade de alteração do software, a empresa solicitou outras duas melhorias ao software. Uma delas é a dificuldade de instalar o sistema em um outro computador, portanto surge um novo requisito, que é facilitar a instalação do software em múltiplos computadores, que será atendido pelas alterações comentadas anteriormente. E a outra melhoria solicitada pela empresa é a possibilidade do software funcionar em sistema operacional Linux, este será atendido pela escolha da tecnologia de desenvolvimento. Os requisitos RF005 e RF013 não serão implementados nesta versão. Isto foi acordado com a empresa devido a indefinição de como armazenar números de série gravados e se é necessário incrementar após a gravação de um produto. O foco desta implementação será na funcionalidade básica, que é gravar os produtos.

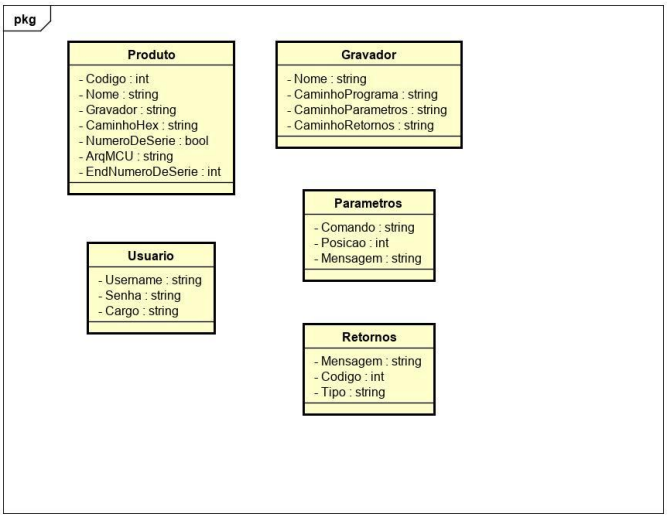
4.5.2 Reprojetar

O processo de engenharia reversa apresentou alguns problemas no projeto do software legado. Podemos citar o fato de boa parte da lógica de negócio ficar dentro das camadas de interface, tentativa de aplicar o padrão de projeto *Data Object Access* mal sucedida, senhas ou localização de arquivos mantidos de forma rígida dentro do código e falta de modularização dos componentes do software. Na fase reprojeto é feita a correção dos erros encontrados para melhor atender as necessidades da empresa. Será preservada a arquitetura do sistema, conforme é recomendado na bibliografia estudada.

A primeira etapa do reprojeto tratou dos dados externos do programa. A ideia dos dados existentes foi mantida, o que mudou foi alguns atributos para compor melhor a nova solução. A figura 18, mostra os dados reprojetados. As principais modificações foram:

- Produto: foi simplificado toda a estrutura, contém somente o básico para realizar a gravação de um produto. A chave primária é o Código e usa uma chave estrangeira que é o nome do Gravador que o produto utilizado. O dado inteiro é gravado no banco de dados.
- Gravador: o caminho do programa do gravador foi retirado do banco de dados. Agora quando é criado um novo gravador o software gera um arquivo de texto contendo o caminho. Este arquivo fica armazenado localmente na máquina que está instalado o software. Os atributos CaminhoParametros e CaminhoRetornos são salvos no banco de dados. Estes caminhos são para arquivos de texto que contém os dados de Parametros e Retornos indicados na figura 18. A ideia destes arquivos é que permaneçam centralizados na rede da empresa. O atributo Nome é a chave primária do dado.
- Parametros e Retornos: como já comentando no item de Gravador, são os dados extraídos de arquivos de texto no formato *XML*. É feito um arquivo para cada gravador cadastrado.
- Usuario: novo dado externo que foi criado para atender as restrições de acesso a certas funcionalidades do programa. Ficará armazenado em banco de dados e possui Username como chave primária. Serão criados dois tipos de usuário, o operador que tem permissão apenas para gravar e o supervisor com permissão total para cadastrar items e gravar.

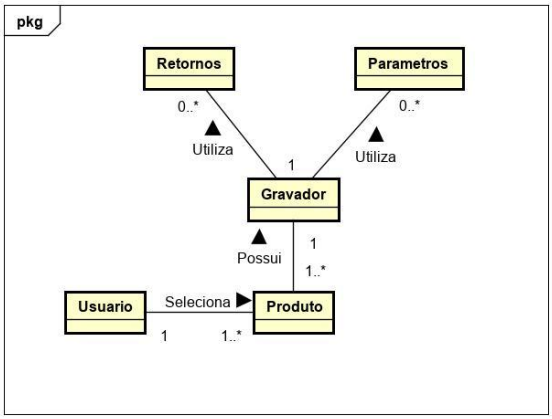
Figura 18: Dados externos reprojetados



Fonte: do autor

Com os novos dados criados cria-se uma nova relação entre estas classes. A nova relação é mostrada na figura 19. As principais diferenças estão no usuário, que passa ter relação com a seleção de produtos e o gravador que utiliza os dados de Retornos e Parâmetros. Antes, o gravador não tinha relação com estes dados.

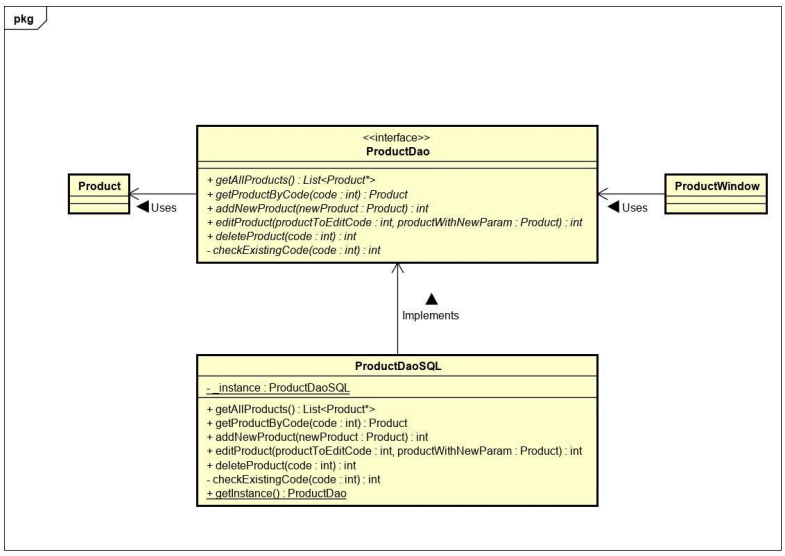
Figura 19: Relação entre os dados externos



Fonte: do autor

Após a redefinição dos tipos de dados, é necessária um novo projeto das classes de negócio que irão trabalhar com os dados externos. A primeira definição feita foi a organização de classes para acesso ao banco de dados. O processo de acesso ao banco de dados implementado tentou seguir o modelo de *Direct Access Object* (DAO), entretanto falharam em sua execução. Na reimplementação será corrigido este padrão de projeto escolhido. A figura 20 mostra o esquema de classes para implementar este padrão para todas as classes de dados externos do programa mostrados na figura 18. A implementação de um *DAO* começa na criação de uma classe de interface, onde contém apenas métodos virtuais. Estes métodos devem ser implementados por uma outra classe, que no caso do projeto será implementado acesso para o banco de dados do tipo *SQL*. Este padrão de projeto esconde da aplicação o tipo de implementação dos dados. Caso o esquema de acesso aos dados mude, será apenas necessário mudar a classe de implementação.

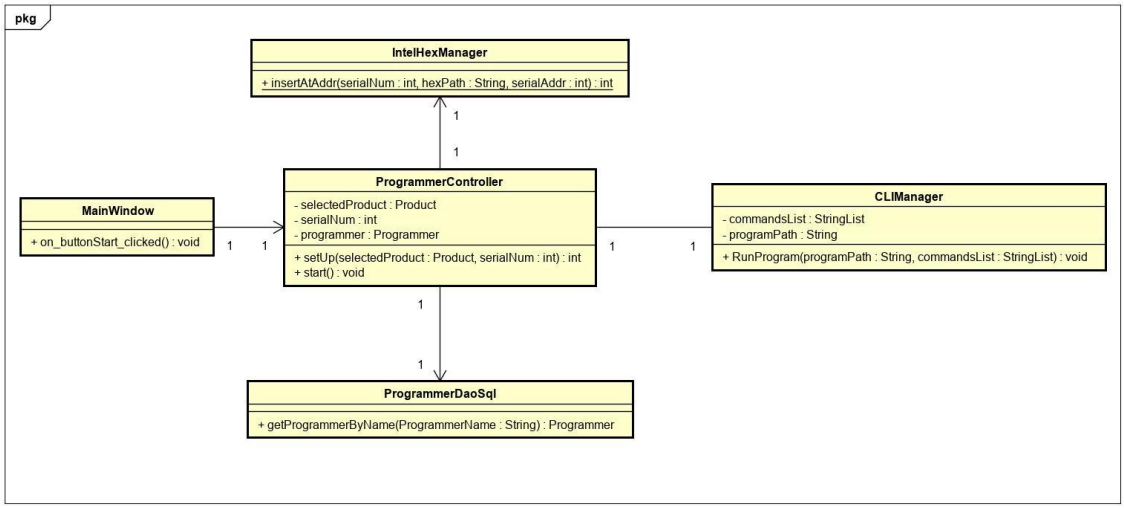
Figura 20: Diagrama de classes *DAO* para dados



Fonte: do autor

O diagrama apresentado na figura 21 foi elaborado para representar as principais classes e métodos projetados para gravar um produto. A chamada de gravação inicia na classe *MainWindow* que implementa a tela principal do programa. Nessa chamada é criada a classe *ProgrammerController* que recebe o produto selecionado e o número de série digitado. Através do produto selecionado, com o parâmetro de nome de gravador, a classe realiza uma chamada para classe de acesso ao banco de dados, *ProgrammerDaoSQL*, que carrega os atributos do gravador. De posse dos atributos de gravador, é feita a inserção de número de série no arquivo hexadecimal através da classe especialista *IntelHexManager*, que tem lógica dedicada para trabalhar com dados hexadecimal no padrão Intel. Na sequência é criada a classe *CLIManager* que executa a lista de comandos apontado pelo gravador no seu programa de gravação e retorna o resultado da operação para classe *ProgrammerController*. Esta por sua vez comunica-se com classe *MainWindow* os resultados de sua operação.

Figura 21: Diagrama de classes processo de gravação



Fonte: do autor

Para auxiliar os processos e modularizar o programa foram criadas classes especialistas. As classes projetadas são:

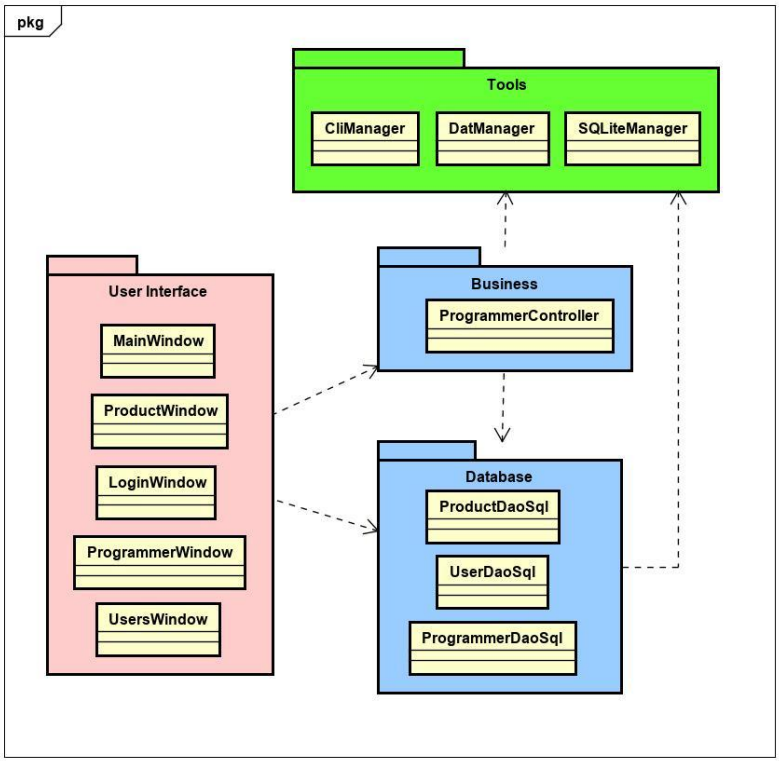
- *IntelHexManager*: classe responsável para inserir o número de série em arquivos hexadecimal no padrão Intel. Mostrada na figura 21;
- *DatManager*: será responsável pelos acessos a arquivos de configuração e dados do programa com extensão *.dat*;
- *CLIManager*: responsável por realizar as chamadas de sub-processos, que são os programas dos gravadores. Mostrado na figura 21;
- *SQLManager*: classe que implementa a inicialização de um banco de dados do tipo *SQL*;

De maneira geral, o projeto do software terá a estrutura mostrada na figura 22. Nela observa-se os quatro grandes componentes que englobam as seguintes funcionalidades:

- *UserInterface*: agrupa as classes que implementam a interface. Possui acesso a classe da lógica de gravação de produto e das classes de Database, pois necessita mostrar para o usuários os dados gravados;

- *Business*: agrupa classes específicas com regras de negócio do sistema. No caso, possui apenas a classe de controle do gravador;
- *Database*: possui a implementação da lógica de acesso aos dados externos do programa.
- *Tools*: conjunto de classes especialistas em áreas diversas que auxiliam outros módulos do sistema.

Figura 22: Estrutura das classes



Fonte: do autor

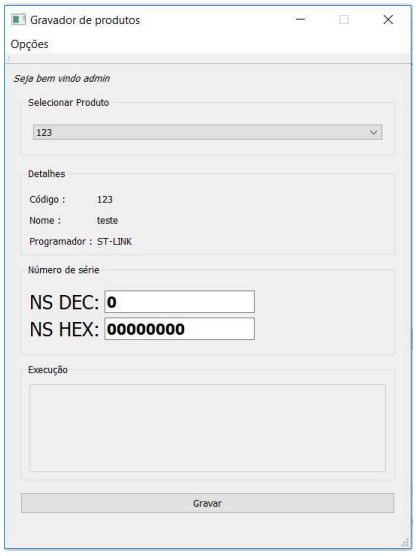
4.5.3 Recodificar

Neste processo foi escolhida uma nova linguagem de programação. Ao invés de seguir utilizando a linguagem `C#`, foi escolhido utilizar a linguagem C++ através do *framework QT*. Esta ferramenta fornece utilitários para desenvolver interfaces de usuário e diversas bibliotecas que facilitam o desenvolvimento de um software. O motivo da escolha é pelo fato de que este *framework* possui licença gratuita, desde que o código fonte das bibliotecas do *framework* sejam disponibilizadas junto com o programa desenvolvido. Outro motivo, é de que é possível compilar o mesmo código para diferentes sistemas operacionais, sem grandes alterações. No caso deste projeto, existe a possibilidade do programa ser executado em ambientes Linux.

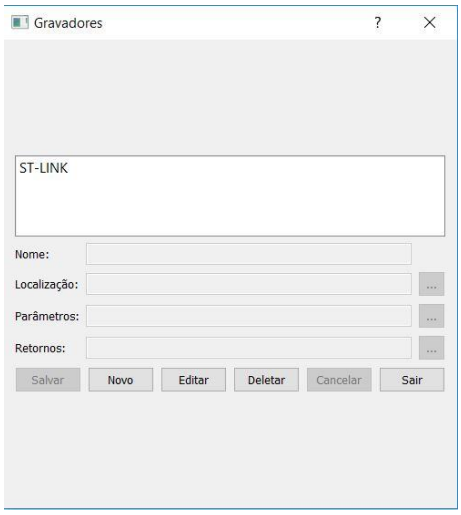
O primeiro passo para recodificar foi a montagem das telas de usuário. O padrão de localização de botões, listas e textos seguiu do software legado. Isto foi feito pensando no usuário que está acostumado com o sistema anterior. As telas elaboradas podem ser

vistas na figura 23. As únicas telas que não existem no software legado é de *login* e a de cadastro de novo usuário.

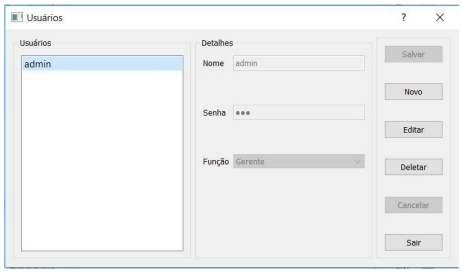
Figura 23: Principais telas do programa



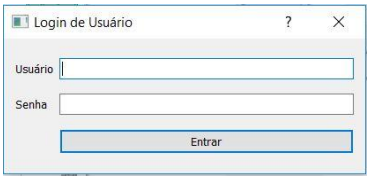
(a) Tela Principal



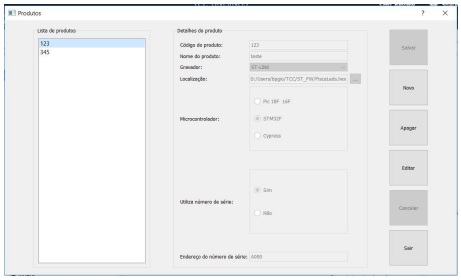
(b) Cadastro de gravadores



(c) Cadastro de usuário



(d) Login



(e) Cadastro de produtos

Fonte: do autor

Após a montagem dos elementos visuais do programa, é iniciada implementação das classes projetadas anteriormente. A programação iniciou pelas classes que compõem a estrutura de acesso ao banco de dados. Cada dado externo do programa foi implementado na estrutura de *Direct Access Object*. Portanto, cada classe dado possui uma *interface* com os métodos virtuais desejados e um classe com a implementação real dos métodos. Em cada classe de implementação, foi codificado uma forma de criação *Singleton*, ou seja, só é possível criar uma instância do objeto de implementação. Todas as formas de

implementação utilizam *SQL* como forma de persistência, com exceção dos dados para Programadores, que mescla persistência em *SQL* e arquivo de texto com extensão *.dat* para salvar e ler dados em formato *XML*. As classes de implementação possuem auxílio de classes especializadas chamadas de *sqlitemanager* para acesso ao banco de dados e *datmanager* para acessar arquivos de texto.

A classe *ProgrammerController* realiza as principais funções do software. Ela é instanciada imediatamente após o clique na tela principal para gravar um produto, através do mecanismo de sinais e *slots* que serão explanados mais adiante. Após seu instaciamento a classe realiza um ciclo de *set-up*, que consiste em:

1. Acessar gravador do produto.
2. Verificar que caminho do *hex* do produto existe.
3. Criar arquivo temporário do *hex* e copiar os dados do arquivo apontado pelo produto.
4. Verificar se é necessário inserir número de série, em caso positivo, realiza-se inserção do número através da classe especialista *intelhexmanager* no arquivo de *hex* temporário.
5. Verifica e carrega o arquivo de retornos apontado pelo gravador em uma lista interna.
6. Verifica e carrega arquivo de parâmetros apontado pelo gravador em uma lista interna.
7. Realiza a serialização dos parâmetros.

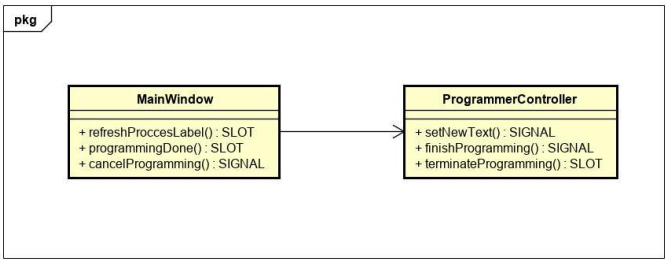
Finalizado o *set-up*, é iniciado o processo de gravação através de um método *Start* da classe. Este inicia um temporizador para controlar o tempo máximo de execução de um programa de gravação e instancia uma classe especialista chamada de *climanager* que recebe o caminho do programa gravador e os dados serializados anteriormente. A classe *climanager* inicia o programa gravador com a classe interna *QProcess*, que fornece todos os métodos de controle do processo externo. A *climanager* realiza comunicação com a *ProgrammerController* através de sinais e *slots*, e é por essa via que ela passa os retornos recebidos pelo processo externo. A execução da *climanager* encerra quando a *ProgrammerController* receber o sinal de retorno indicando fim de gravação ou quando o temporizador acaba.

O *framework QT* fornece um sistema de comunicação entre objetos chamado de sinais e *slots*. Este sistema é uma alternativa a técnica de *callbacks*. De forma resumida, ao ocorrer um evento como um botão clicado, um sinal é gerado pelo objeto que controla esse botão. Um *slot* é a função a ser chamada quando um sinal é emitido. No projeto esse sistema foi bastante utilizado para realizar a comunicação entre objetos, principalmente para definir as ações de alguns botões da interface e outras ações do usuário.

Para exemplificar e mostrar o uso, foi escolhido o processo gravação de um produto. Para ilustrar foi feito um diagrama mostrado na figura 24 todos os sinais e *slots* criados pelas classes envolvidas no processo. A classe *MainWindow* possui um *slot* chamado *refreshProccesLabel* para receber textos do processo de gravação e um outro *slot* chamado de *programmingDone* para ser informado que a classe que controla a gravação encerrou a atividade. Além disso, a classe *MainWindow* possui um sinal para emitir para a classe de controle de gravação, chamado de *cancelProgramming*, que é emitido quando o usuário tentou encerrar a gravação.

Do lado da classe que controla o processo de gravação, a *ProgrammerController*, emite sinal *setNewText* indicando que tem um novo texto para atualizar e o sinal *finishProgramming* para sinalizar que seu processo encerrou. Possui ainda um *slot* para terminar o processo de gravação.

Figura 24: Classes com sinais e *slots*



Fonte: do autor

5. Validação

O seção final do trabalho tem o objetivo de apresentar o processo de validação que o sistema passou. Inicialmente o software foi validado internamente pelo pesquisador para testar se os requisitos implementados atendem as funcionalidades desejadas. Em seguida, foi montado uma *release* do programa para empresa e, através de entrevistas, foi feito um acompanhamento da situação dos testes.

5.1. Testes internos

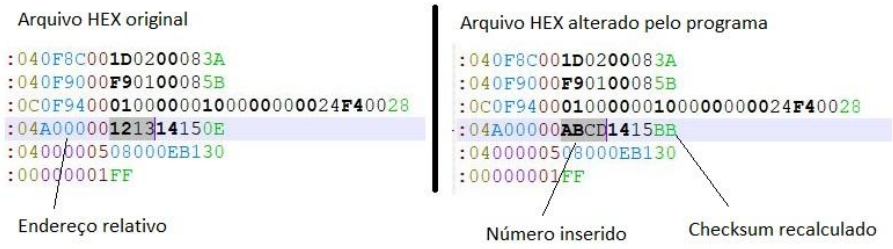
O objetivo de realizar os testes internos é para garantir que o sistema esteja atendendo os requisitos estabelecidos, tentando evitar problemas na validação em campo. Foi desenvolvido um roteiro de testes com a descrição dos requisitos e método de teste, o resultado esperado e o resultado obtido. Caso o resultado obtido fique fora do esperado é realizada a correção no componente que apresentou defeito.

O roteiro estabelecido foi executado dois dias antes da liberação da primeira *release* e foram necessárias duas horas para validar todos os requisitos listados.

5.1.1 Roteiro de testes internos

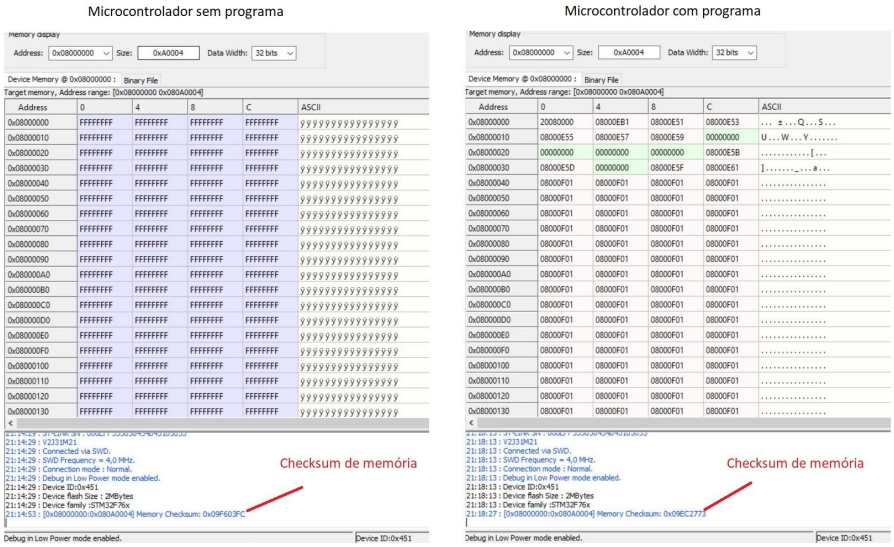
- **Requisitos:** RF001, RF002, RF003 e RF004
  - Descrição: testar todos os métodos CRUD para os dados externos. Verificar integridade dos dados no banco.
  - Resultado obtidos: todos dados cadastrados atributos com íntegros.
- **Requisito:** RF006
  - Descrição: inserir o número hexadecimal 0xABCD no endereço 0xA000 apontado pelo arquivo hexadecimal.
  - Resultado obtido: número inserido com sucesso, ver figura 25.
- **Requisito:** RF007
  - Descrição: gravar um microcontrolador através do programa.
  - Resultado obtido: microcontrolador gravado com sucesso, ver comparativo de memória com e sem gravação na figura 26.

Figura 25: Inserção de número de série em arquivo hexadecimal



Fonte: do autor

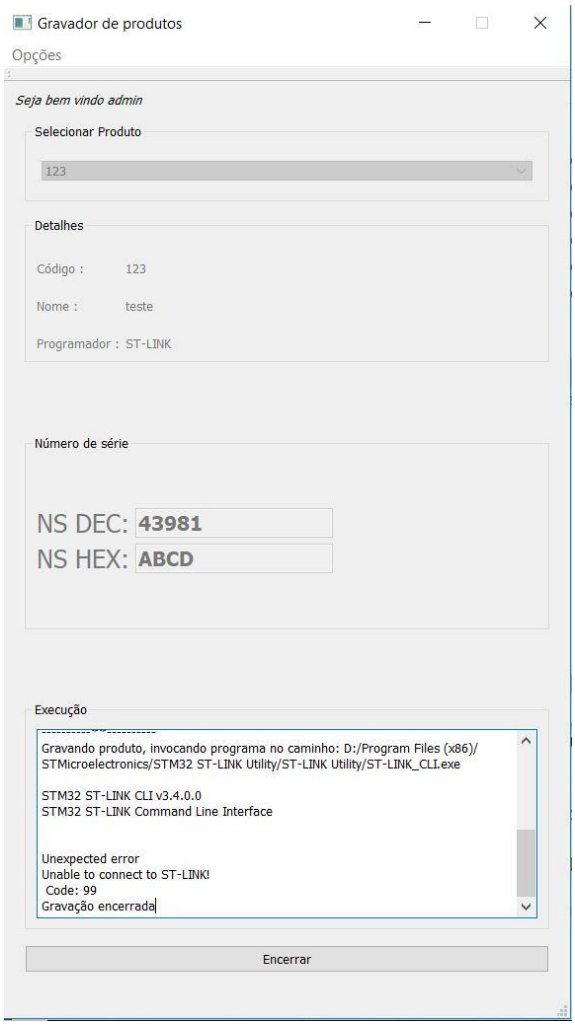
Figura 26: Gravação de microcontrolador



Fonte: do autor

- **Requisito:** RF008
  - Descrição: disponibilizar erros do processo de gravação para o operador.
  - Resultado obtido: *log* mostrando todos os passos e resultados obtidos, a figura 27 mostra a tela principal apresentando o resultado.
- **Requisito:** RF009
  - Descrição: bloquear usuários do tipo Operador no acesso a telas de cadastro.
  - Resultado obtido: o usuário operador não consegue acessar as telas de cadastro.
- **Requisito:** RF010
  - Descrição: pesquisar um produto na barra de seleção de produto através de seu código. de produtos.
  - Resultado obtido: barra de seleção de produto com os códigos.
- **Requisito:** RF011
  - Descrição: disponibilizar as informações solicitadas na tela principal do programa.
  - Resultado obtido: todas as informações solicitadas são mostradas ao selecionar um produto.

Figura 27: Tela principal com *log* de gravação



Fonte: do autor

- **Requisito:** RF012
  - Descrição: ser possível digitar número em formato decimal quanto hexadecimal
  - Resultado obtido: é possível digitar um número decimal em um campo e hexadecimal em outro.

5.2. Testes externos

Com a fase de testes internos concluída, o software foi enviado para a Comlink iniciar seu uso na data 02/05/2019. Para o primeiro uso foi elaborado pelo pesquisador, um manual de instruções de como ajustar o banco de dados e os parâmetros de gravador. Este manual foi enviado para empresa, junto com o arquivo executável do programa, que foram disponibilizados em uma pasta compartilhada do *Google Drive*. A empresa utilizou o software durante o dia e ao final mandaram um lista de problemas encontrados durante o uso através do contato de entrevista por *e-mail*. Nessa primeira *release*, cada problema informado foi analisado e a devida solução foi informada.

5.2.1 Primeira *release*

A lista de problemas e melhorias sugeridas nesse primeiro momento foi a seguinte:

- **Problema**: Em determinados computadores a interface ficou muito grande, com elementos desaparecendo na barra de tarefas do Windows;
  - **Tipo**: Melhoria;
  - **Solução**: Tamanho da janela principal estava sem valor fixo, causando distorção em determinados momentos. Valores de tamanhos máximos foram fixados;
- **Problema**: Quando algum item é cadastrado, a mensagem de sucesso aparece com símbolo de ”WARNING”;
  - **Tipo**: Melhoria;
  - **Solução**: Trocar a função *QMessageBox::warning* para *QMessageBox::information* quando um item é cadastrado com sucesso;
- **Problema**: Não foi possível gravar um produto com gravador da ST-LINK e nem inserir número de série;
  - **Tipo**: *Bug*;
  - **Solução**: Foi identificado que o caminho do arquivo hexadecimal deste produto possuía caracteres especiais como acento em palavras e a letra ’ç’. O programa inicialmente gravava o caminho corretamente no banco de dados, entretanto ao ler os dados do banco, o software decodificava os caracteres especiais de maneira errada. Portanto ao iniciar o programa foi necessário montar a seguinte função:

    ```
    QTextCodec::setCodecForLocale(QTextCodec::
    ↪ codecForName("utf8"));
    ```

    O compilador do QT utiliza como padrão o *codec* LATIN-1, a função troca o *codec* para UTF8;
- **Problema**: Na edição de senha do usuário são mostrados os caracteres reais da senha, necessário esconder por algum símbolo;
  - **Tipo**: Melhoria;
  - **Solução**: Foi utilizado um método do próprio criador de interface do programa QT que troca os caracteres por ”*”;

O problema relatado como *bug* travou por completo o andamento dos testes. Por isso a correção para retomada dos testes foi feita de forma urgente pelo pesquisador.

5.2.2 Segunda *release*

Após as correções citadas acima, foi realizado também uma melhoria nos *logs* de gravação, com a inclusão de não apenas os retornos que o software do gravador está mandando, mas também os passos que o Gravador de Produtos está executando, para assim ficar mais fácil de identificar algum problema.

A segunda rodada de teste teve início na data 17/05/19, com retorno de resultado no dia 24/05/19 e foram relatados dois problemas. O primeiro problema corresponde aos produtos com gravador ST-LINK não estavam gravando. Analisando o *log* da gravação, foi identificado o motivo do problema. Abaixo está a linha que informa onde está sendo gerado o arquivo temporário hexadecimal:

```
Hex temporario criado em: C:/Users/matheus.COMLINK/Downloads
    ↪ /release (1)/Temp/tempHex.hex
```

O *log* informa também como foram serializados os comandos que serão passados para o processo gravador:

```
Comandos serializados:

-ME

-P C:/Users/matheus.COMLINK/Downloads/release (1)/Temp/
    ↪ tempHex.hex

-rst

Fim setup
```

O comando **-P** carrega o caminho do arquivo hexadecimal e neste caminho temos uma pasta chamada de **release (1)**. O programa do ST-LINK ao receber este caminho, não conseguia interpretar a linha devido esta pasta possuir no nome o símbolo de parenteses. Por causa disso o programa retornava ao operador que o formato do arquivo de gravação estava inválido.

Diante desse problema, foi feita uma orientação para trocar o nome da pasta para não possuir nenhum caractere de parenteses, pois ao que tudo indica, é problema do software específico do hardware de gravação, e não do Gravador de Produtos.

O segundo problema encontrado aconteceu durante os testes do software no computador utilizado pela produção. Ocorreu que a gravação em alguns produtos estava levando mais tempo que o normal o que acabava estourando o temporizador que aguarda algum tipo de retorno do processo de gravação. Esse problema ocorre especialmente em gravadores ST-LINK, pois este só faz o retorno quando encerra a gravação. O que foi feito para resolver este problema foi aumentar o temporizador. No momento não foi possível identificar a causa do problema, pois este acontece de maneira esporádica.

Foi montada uma terceira *release* do software para a Comlink. Esta encontra-se em testes no ambiente de produção e está em uso paralelo com o software legado. Isto está

sendo feito para avaliar a robustez do sistema e procurar falhas que nem sempre aparecem em testes normais.

6. Conclusão

Este trabalho teve por objetivo avaliar se técnica de reengenharia é uma boa alternativa quando é desejado evoluir um software de alta importância para uma organização, que no caso deste estudo foi a Comlink Equipamentos Eletrônicos. O software em questão era utilizado no processo de gravação de produtos eletrônicos e apresentava problemas para evoluir e acompanhar os novos anseios da empresa.

Através do estudo realizado na seção 2 pode-se entender melhor as técnicas e metodologias utilizadas para reengenharia, além de suas tecnologias como a engenharia reversa. Toda a fundamentação teórica fornecida por Osborne e Chikofsky (1990) e Waters e Chikofsky (1994), ajudou no entendimento dos conceitos e palavras chaves utilizadas dentro do contexto de reengenharia. Os trabalhos analisados de Rosenberg (1996) e Pressman (2016) tiveram uma importante contribuição para abordagem e fluxo de trabalho para a aplicação de reengenharia nesta pesquisa.

A seção quatro foi onde concentrou-se o maior esforço deste trabalho. O processo de engenharia reversa a partir do software implementado e seu código legado conseguiu abstrair os requisitos que outrora foram elaborados para o desenvolvimento do software legado. A atividade consistiu em observar o funcionamento do programa, analisar o código fonte, as tecnologias utilizadas no desenvolvimento e as interfaces criadas. Com os dados da engenharia reversa, deu-se início ao processo de engenharia avante. Os requisitos relacionados a salvar número de série gravado e incremento automático de número de série foram retirados para esta versão, pois não há até o momento uma estratégia para automatizar a geração de números de série por parte da empresa. Houve também a inserção de novas funcionalidades, como autenticação de usuário e a possibilidade de editar o caminho do banco de dados. O objetivo destas modificações foram para promover logo uma disruptura se comparado ao software legado.

Na última parte deste trabalho foi realizada uma bateria de testes internos e externos. Internamente, foi feito testes para saber se os requisitos estipulados estavam sendo cumpridos. Em seguida, foram feitos testes na empresa. A primeira *release* foi a que mais apresentou problemas, entretanto, apenas um foi considerado um *bug* e este não era relacionado aos requisitos. Foi feita uma segunda *release* e nesta foi necessário um ajuste no tempo máximo sem resposta que um processo de gravação pode durar. Portanto, considera-se que o software reimplementado, em termos de funcionalidades, está cumprindo o seu papel dentro da organização

Através deste trabalho pode-se confirmar que aplicar a reengenharia foi uma boa alternativa para software Gravador de Produtos. Neste processo de observar e analisar o sistema legado, conseguimos extrair as ideias iniciais do projeto e ver aquilo que funcionou e não funcionou na implementação do software. No caso deste projeto, percebeu-se que por exemplo, armazenar os números de série não era algo útil atualmente para a organização e que seu incremento automático também não tinha nenhuma utilidade. Este processo permite também entender as falhas de desenvolvimento que aconteceram, como requisitos mal interpretados e funcionalidades mal implementadas. Este conhecimento ajuda a guiar o caminho da reimplementação, mesmo quando o desenvolvedor original do

projeto não esteja mais acessível ou mesmo que não exista documentação do software.

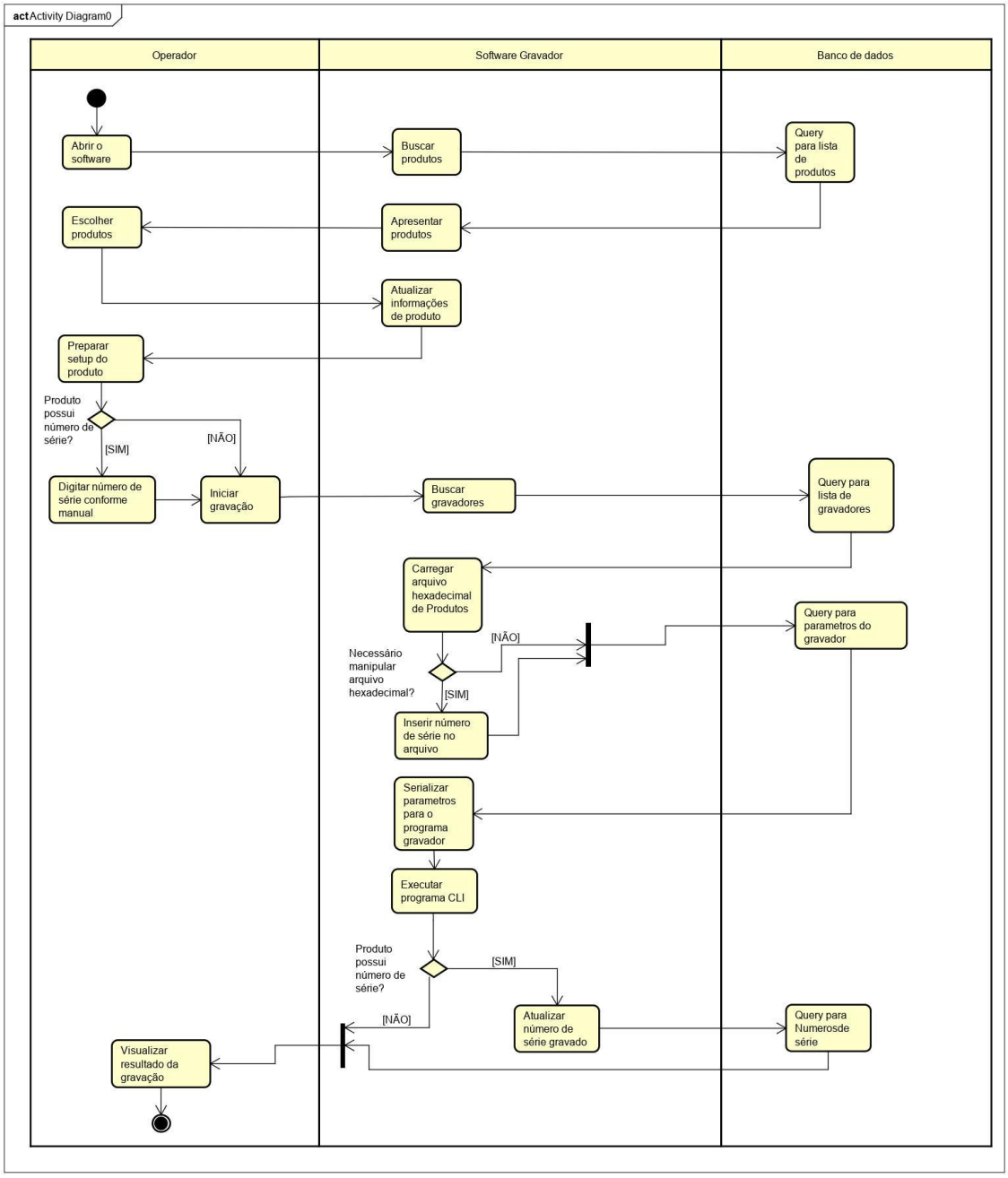
Entretanto, muitas das análises executadas na fase de engenharia reversa foram manuais. O código foi revisado todo manualmente, com um pouco de ajuda de uma ferramenta que analisa dependências do código e do próprio ambiente de desenvolvimento do software. Este processo é lento, pois é necessário bastante interpretação de código para abstrair as informações que estavam somente na cabeça de quem desenvolveu o software legado. E também não foi possível utilizar reaproveitamento de código, devido ao alto acoplamento que existia na implementação antiga. Esses detalhes atrasaram o desenvolvimento, e se fosse em um projeto de maior dimensão e complexidade, o atraso poderia ser muito grande. Por isso fica evidente que a reengenharia possui etapas extras ao ser comparado com um novo desenvolvimento, que por vezes podem comprometer os prazos de entrega do sistema reimplementado. Além de tentar sempre mitigar riscos como inclusão de novos requisitos e funcionalidades, deve-se tentar negociar a inclusão destes para após a reimplementação estar devidamente validada. Novos requisitos podem inserir novas chances de erro no sistema que podem comprometer todo o projeto.

De maneira geral, os objetivos deste trabalho foram alcançados. Pode se verificar que reengenharia é sim uma boa alternativa para recuperar e renovar um software, devido a facilidade que os requisitos foram recuperados e devidamente reimplementados em nova forma. Quanto aos específicos, todos foram igualmente alcançados. O objetivo de aplicar reengenharia baseado na bibliografia transcorreu ao longo do capítulo 4, juntamente com o objetivo de implementar o sistema baseado na modelagem obtida, que é descrito a partir do item 4.5. A validação acontece em todo capítulo 5, e foi totalmente baseada nos *feedbacks* da empresa Comlink, que é a principal interessada neste sistema.

Como proposta de trabalho futuro, é sugerido uma pesquisa para avaliar se se o software reimplementado por este estudo apresentou alguma melhora significativa no processo de adicionar funcionalidades e aplicar manutenções dentro da empresa Comlink. Sugere-se ainda utilizar este estudo como comparativo para outros métodos de evolução e manutenção de software.

7. Apêndices

A. Fluxograma de atividade de gravação abstraído

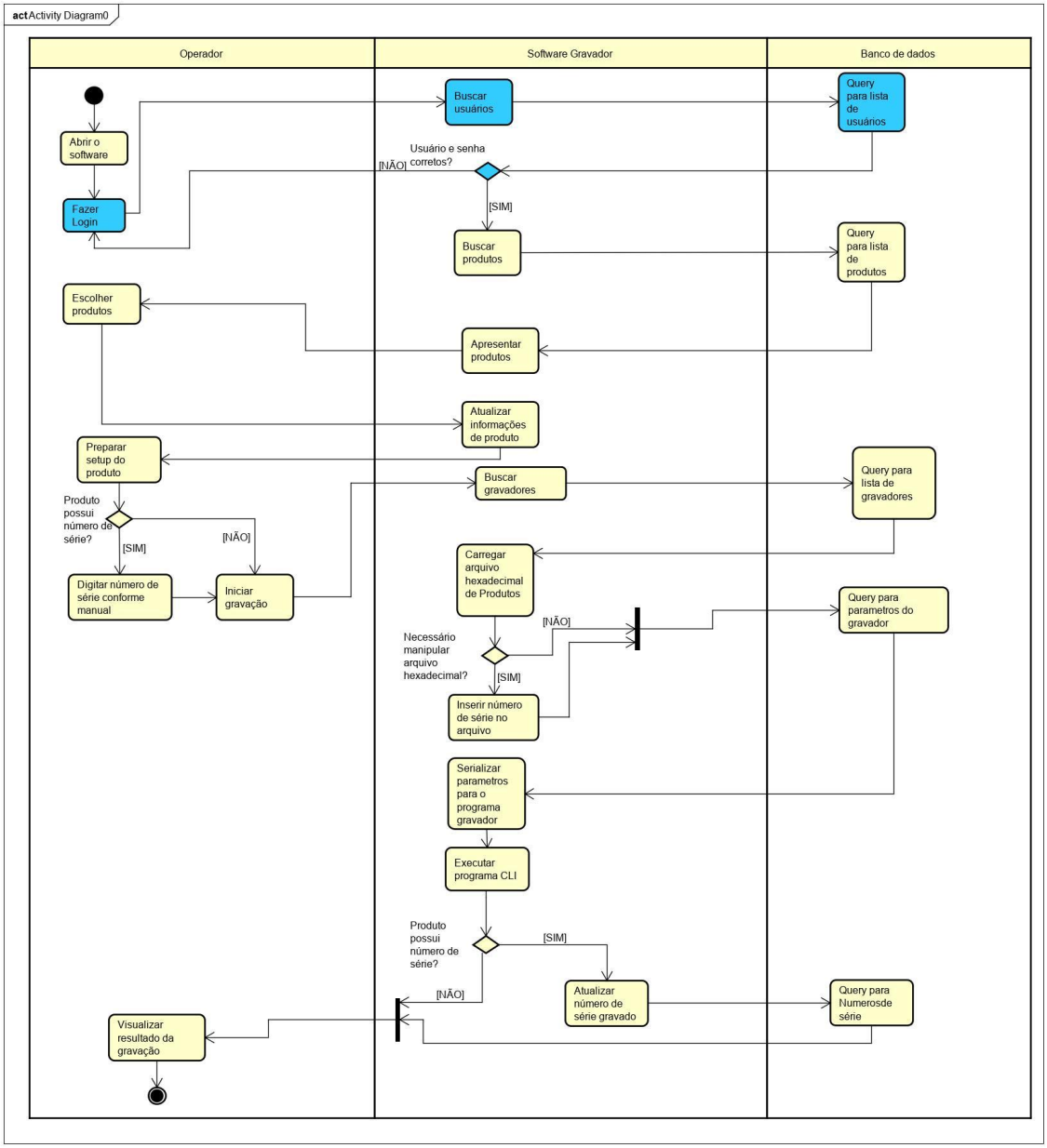


B. Dados de parâmetros e retornos salvos em arquivos

```
<Parameters>
 <Programmer>
  <Name>PicKit </Name>
  <Step>1</Step>
  <Message>Erase </Message>
  <Command>-P</Command>
 </Programmer>
  <Programmer>
  <Name>PicKit </Name>
  <Step>2</Step>
  <Message>Detect </Message>
  <Command>-L1</Command>
 </Programmer>
  <Programmer>
  <Name>PicKit </Name>
  <Step>3</Step>
  <Message>Detect </Message>
  <Command>-M</Command>
 </Programmer>
  <Programmer>
  <Name>PicKit </Name>
  <Step>4</Step>
  <Message>Program </Message>
  <Command>-F%H</Command>
 </Programmer>
</Parameters>

<ReturnCodes>
 <Programmer>
  <Name>PicKit </Name>
  <Code>0</Code>
  <Message>Program Succeeded</Message>
  <Type>Sucesso </Type>
 </Programmer>
 <Programmer>
  <Name>PicKit </Name>
  <Code>36</Code>
  <Message>Parametros para gravacao invalidos</Message>
  <Type>Erro </Type>
 </Programmer>
 <Programmer>
  <Name>PicKit </Name>
  <Code>39</Code>
  <Message>Auto-Detect: No known part found.</Message>
  <Type>Erro </Type>
 </Programmer>
</ReturnCodes>
```

C. Fluxograma alterado



8. Referências

Referências

ALMEIDA, R. F. de. *Aplicação Prática de Técnica de Reengenharia de Software*. 65 p. Monografia (Ciência da Computação) — Universidade Federal de Juiz de Fora, Juiz de Fora, 2009.

ARM. *ARM Debug Interface Architecture Specification*. [S.l.], 2017.

ARNOLD, R. S. Software restructuring. In: *Proceedings of the IEEE*. [S.l.]: IEEE, 1989. v. 77, p. 607 – 617.

AZEVEDO, D. et al. *Métodos e procedimentos de pesquisa: do projeto ao relatório final*. [S.l.]: Editora Unisinos, 2011.

BARDIN, L. *Análise de conteúdo*. Lisboa: Persona, 1995.

BERNHARTR, M. et al. Incremental reengineering and migration of a 40 year old airport operations system. *28th IEEE International Conference on Software Maintenance (ICSM)*, 2012.

CAGNIN, M. I. *PARFAIT: uma contribuição para a reengenharia de software baseada em linguagens de padrões e frameworks*. Tese (Doutorado) — Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo, São Carlos, sep 2005.

CHIKOFSKY, E.; CROSS, J. Reverse engineering and design recovery: a taxonomy. *IEEE Software*, v. 7, p. 13 – 17, 1990.

EASTERBROOK, S. et al. Selecting empirical methods for software engineering research. In: *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. [S.l.]: IEEE, 2007. v. 77, p. 574–574.

EILAM, E. *Reversing: Secrets of Reverse Engineering*. Indianapolis, Indiana: Wiley Publishing, Inc., 2005.

FLICK, U. *Qualidade na Pesquisa Qualitativa*. Porto Alegre: Artmed, 2009.

MüLLER, H. A. et al. Understanding software systems using reverse engineering technology perspectives from the rigi project. *CASCON '93 Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering*, v. 1, p. 217–226, 1995.

OSBORNE, W. M.; CHIKOFSKY, E. J. Fitting pieces to the maintenance puzzle. *IEEE Software*, p. 11–12, 1990.

PARNAS, D. L. Software aging. In: *16º Conferência de Engenharia de Software*. Sorrento - Itália: IEEE Computer Society Press, 1994. p. 279–287.

PIEKARSKI, A. E. T.; QUINAIA, M. A. Reengenharia de software: o que, por quê e como. *Revista Ciências Exatas e Naturais*, v. 1, 2000. Acessado em 14 de abril de 2019, disponível em : https://revistas.unicentro.br/index.php/RECEN/article/view/528.

PRESSMAN, R. S. *Engenharia de software : uma abordagem profissional*. 8º. ed. [S.l.]: AMGH Ed, 2016.

RICHARDSON, R. J. *Pesquisa social: métodos e técnicas*. 3º. ed. São Paulo: Atlas, 1999.

ROSENBERG, L. H. Software re-engineering. *Technical Report SATC-TR-95-1001, NASA*, 1996. Https://pdfs.semanticscholar.org/d6b2/89019f9fef924fd3300d1094f29c8bc079f9.pdf. Acessado em 12 de maio de 2019.

SINGER, J. et al. Software engineering data collection for field studies. *Guide to Advanced Empirical Software Engineering*, Springer, p. 9–34, 2008.

SOMMERVILLE, I. *Engenharia de Software*. 6º. ed. São Paulo: Addison-Wesley, 2004.

SOMMERVILLE, I. *Engenharia de Software*. 9º. ed. São Paulo: Pearson, 2011.

SOUSA, H. P.; LEITE, J. C. S. do P. *Aplicação da Engenharia Reversa e Reengenharia de Software no Desenvolvimento de Plugins para a Ferramenta Oryx*. 64 p. Monografia (Ciência da Computação) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2012.

STMICROELECTRONICS. *UM0892: STM32 ST-LINK utility software description*. [S.l.], 2018.

WATERS, R. C.; CHIKOFSKY, E. Reverse engineering progress along many dimensions. *Communications of the ACM*, v. 37, n. 5, p. 23–24, 1994.